String

Uma palavra representada dentro do computador nada mais é do que uma **sequência de caracteres**, ou melhor dizendo, um *array* de caracteres. Em linguagens de baixo nível, como o C por exemplo, os caracteres precisam ser manipulados individualmente a fim de constituir as palavras e textos dentro de um programa. Em linguagens de alto nível como o Java, este trabalho não é necessário, pois existe a abstração chamada de ***String***.

As *Strings* são uma forma fácil, no entanto extremamente poderosa, de lidar com um conjunto de caracteres. Com essa estrutura em mãos, fica muito fácil criar, copiar, deletar, juntar ou separar palavras. Esta estrutura está longe de ser algo exclusivo do Java e muitas das funções disponíveis para esta classe n

linguagem também estão disponíveis em outras linguagens, como C#, JavaScript e Python, somente para citar algumas.

O uso desta estrutura no Java se faz através da classe *String*. Sua declaração é bastante simples, seguindo a estrutura abaixo.

```
String texto = “Java é muito legal!”; // ou new String(“Java é muito legal!”)
```

Todo o conteúdo inserido entre as aspas é armazenado dentro da variável e associado ao tipo. O uso do construtor (*new String()*) não se faz necessário pois o Java já faz essa inferência por se tratar de um tipo de dado muito utilizado. Inclusive, não é recomendado que se use os construtores, pois a arquitetura do Java já otimiza as *Strings* para otimizar o uso da memória.

Com uma String em mãos, existem alguns métodos importantes de se conhecer.

Tipo de retorno	Assinatura do método	O que ele faz?
char	charAt(int i)	Retorna o caractere correspondente a posição indicada no parâmetro da função.
String	concat(String s)	Concatena, ou seja, junta duas Strings em uma única.
boolean	contains(String s)	Retorna verdadeiro se a String indicada no parâmetro estiver contida na String que invocou o método.
boolean	equals(String s)	Retorna verdadeiro se duas Strings forem “exatamente” iguais.
boolean	equalsIgnoreCase(String s)	Retorna verdadeiro se duas Strings forem “exatamente” iguais , ignorando a diferença entre letras maiúsculas e minúsculas.
int	indexOf(String s)	Retorna o índice da primeira ocorrência

		da String enviada em parâmetro dentro da String que invocou o método. O valor -1 é retornado caso nada seja encontrado.
int	length()	Retorna a quantidade de caracteres de uma String, ou seja, o seu tamanho.
String	toLowerCase()	Retorna a String com todos os caracteres em minúsculo.
String	toUpperCase()	Retorna a String como todos os caracteres em maiúsculo.
String	trim()	Retorna a String com os espaços em branco do início e do fim removidos.
String	substring(int i, int f)	Retorna a substring formada pelos caracteres iniciados no índice indicado pelo primeiro parâmetro e finalizado no índice indicado no segundo parâmetro.
String	valueOf(tipo t) tipo* float, double, int ou long	Retorna a representação em String do tipo indicado como parâmetro.

A classe *String* possui mais de cinquenta métodos que podem ser utilizados pela pessoa programadora para conseguir alcançar seus objetivos na construção de um algoritmo. No entanto, um ponto em especial em que é

necessário ter muita atenção é no aspecto de **comparação de igualdade entre Strings.**

É muito comum na programação os símbolos de comparação serem utilizados para validar se dois valores de um mesmo tipo são iguais. Por exemplo, para validar se o número dois (2) e três (3) são iguais, é possível usar o seguinte código:

```
if(2 == 3) {
  System.out.println(“São iguais!");
} else {
  System.out.println(“São diferentes!);
}
```

Entretanto, quando se trata de classes no Java, no caso específico a classe *String*, essa comparação pode trazer problemas. Para demonstrar, considere o seguinte código:

```
String string1 = "s";
String string2 = "s";
if(string1 == string2) {
  System.out.println("São iguais!");
} else {
  System.out.println("Não são iguais!");
}
```

Ao rodar este código, o resultado será verdadeiro (*true*), ou seja, as *Strings* armazenadas em variáveis diferentes (no caso, *string1* e *string2*) são iguais. Mas uma simples alteração na forma de criar a variável tornará o resultado falso (*false*), apesar das strings serem, a princípio, exatamente iguais.

```
String string1 = "s";
String string2 = new String("s");
if(string1 == string2) {
```

```
  System.out.println("São iguais!");
} else {
  System.out.println("Não são iguais!");
}
```

Por que isso acontece? Por se tratar de uma classe utilizada em abundância nos programas, os projetistas do Java fizeram com que as *Strings* possam ser compartilhadas em memória (*pool*). Ou seja, em muitas ocasiões duas variáveis diferentes poderão estar apontando para um mesmo endereço de memória, pois o conteúdo é o mesmo. No entanto, o operador de comparação (==) usa as **referências de memória** para comparar o texto armazenado na *String*. Se estes endereços são diferentes mas o conteúdo é igual, não importa

o resultado será falso. Por isso, a melhor forma de comparar *Strings* (e outros tipos de dados complexos) é utilizando o método *equals()*.

O método *equals()* utilizado para comparar *Strings* deriva da classe *Object* e é utilizado para testar a relação de igualdade entre duas instâncias de uma mesma classe. Este método está presente em todas as classes e pode ser sobrescrita. Na classe *String* ela foi implementada para comparar somente o conteúdo, sem levar em consideração o endereço de memória. Desta forma, as comparações são assertivas, diferente do que acontece com o operador de comparação. O exemplo abaixo demonstra a maneira ideal de comparar os conteúdos de duas *Strings*.

```
String string1 = "s";
String string2 = new String("s");
if(string1.equals(string2)) {
  System.out.println("São iguais!");
} else {
  System.out.println("Não são iguais!");
}
```

Atividade Extra

- Artigo: Strings no Java - Não Use o Operador de Igualdade!
- Site: Java Online Compiler
- Site: Aprender Java através de jogos

Referência Bibliográfica

BARNES, D. J.; KOLLING, M. **Programação orientada a objetos com java: uma introdução prática usando o bluej.** 4.ed Pearson: 2009.

FELIX, R. (Org.). **Programação orientada a objetos.** Pearson: 2017.

MEDEIROS, L. F. de. **Banco de dados: princípios e prática.** Intersaberes: 2013;

ORACLE. Java Documentation, 2021. **Documentação oficial da plataforma Java.** Disponível em: <https://docs.oracle.com/en/java/>. Acesso em 12 de Jul. de 2021.

Ir para questão