

06.Strings em Java

Prof. Alexandre Krohn



Roteiro

- Utilização de Strings em Java
- Métodos principais da classe String
- Exercícios




Strings

- Uma String é um objeto que armazena uma sequência de caracteres
- Tratamos String como sendo um tipo de dados, mas é um objeto da classe String



Strings são Imutáveis

- Um objeto String é imutável, o que significa que o texto que ele carrega nunca é alterado.
- Sempre que um texto precisa ser modificado é utilizado mais espaço em memória para que uma nova String seja criada contendo a nova versão dele



Strings : Criação

- Para o compilador, qualquer texto entre aspas duplas é uma String.
- Por esse motivo a criação de um objeto desse tipo não requer a utilização do operador new.
- Assim, uma String é criada de forma semelhante a um tipo primitivo, utilizando-se a sintaxe :
`[tipo] [nome] = [valor];`
- apesar de se tratar de um tipo por referência - um nome para um objeto em memória.



Strings : Criação

```
String texto = "Qualquer texto entre aspas é uma String";
```

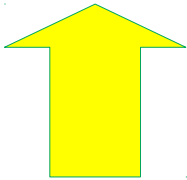
ou

```
String texto = new String("Qualquer texto entre aspas é uma String");
```



Strings : Criação

```
String texto = "Qualquer texto entre aspas é uma String";
```



Este código executado
1000 vezes em um loop criará
apenas 1 objeto String. Java
reaproveita as Strings

ou

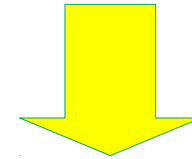
```
String texto = new String("Qualquer texto entre aspas é uma String");
```

Strings : Criação

```
String texto = "Qualquer texto entre aspas é uma String";
```

ou

Este código executado
1000 vezes em um loop criará
1000 objetos **String**. A palavra
chave **new** forçará a criação dos
objetos



```
String texto = new String("Qualquer texto entre aspas é uma String");
```


Strings : Qualquer texto é uma String

- Qualquer texto é uma String, e por isso podemos invocar os métodos da classe String.
- Ex.:

`"Qualquer texto entre aspas é uma String".length();`

O método ***length()*** retorna o comprimento (**número de caracteres**) que a String contém.

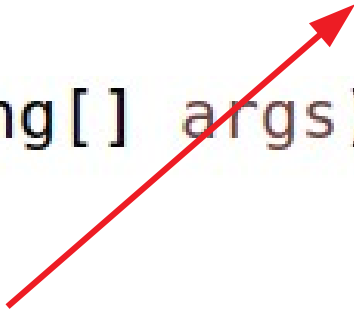
Strings : valor nulo

- Uma variável String não inicializada contém nulo (*null*)

```
class Pessoa {  
    String nome;  
}
```

Ao executar o main,
erá mostrado o valor *null*

```
public static void main(String[] args) {  
  
    Pessoa p = new Pessoa();  
    System.out.println(p.nome);  
  
}
```





Strings : Testando igualdade

- A comparação entre Strings utilizando o operador de igualdade (`==`) retornará true se as duas referências apontarem para o mesmo objeto na memória.



Strings : Testando igualdade

```
String nome = "Arthur";  
String apelido = nome;
```

```
if(nome == apelido) {  
    System.out.println("Nome e apelido são iguais");  
}
```

Strings : Testando igualdade

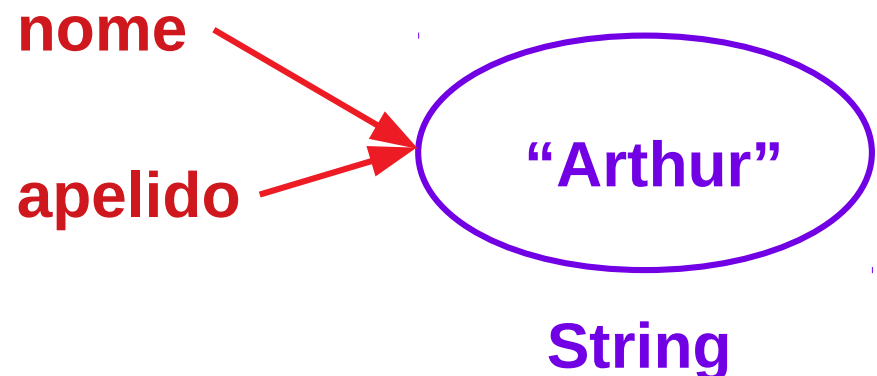
```
String nome = "Arthur";  
String apelido = nome;  
  
if(nome == apelido) {  
    System.out.println("Nome e apelido são iguais");  
}
```

Quando uma variável de objeto
(sua referência) é atribuída a outra,
dizemos que as **duas “apontam”**
para o mesmo objeto.
Por isso o teste retorna verdadeiro

Strings : Testando igualdade

```
String nome = "Arthur";  
String apelido = nome;  
  
if(nome == apelido) {  
    System.out.println("Nome e apelido são iguais");  
}
```

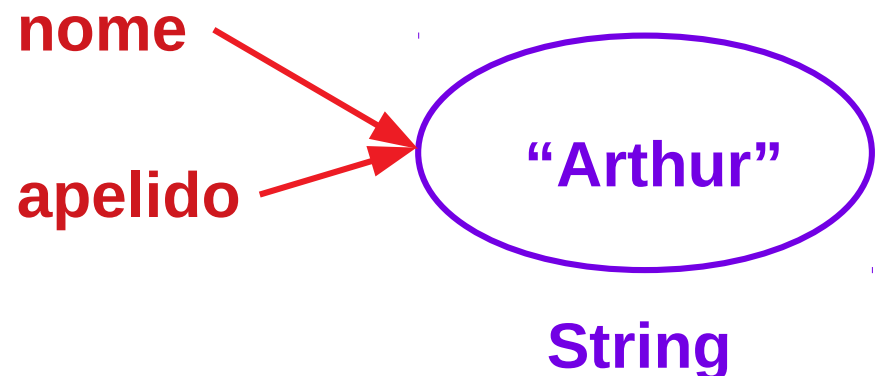
Quando uma variável de objeto (sua referência) é atribuída a outra, dizemos que as **duas "apontam" para o mesmo objeto**. Por isso o teste retorna verdadeiro



Strings : Igualdade (de novo)

```
String nome = "Arthur";  
String apelido = "Arthur";  
  
if (nome == apelido) {  
    System.out.println("Nome e apelido são iguais");  
}
```

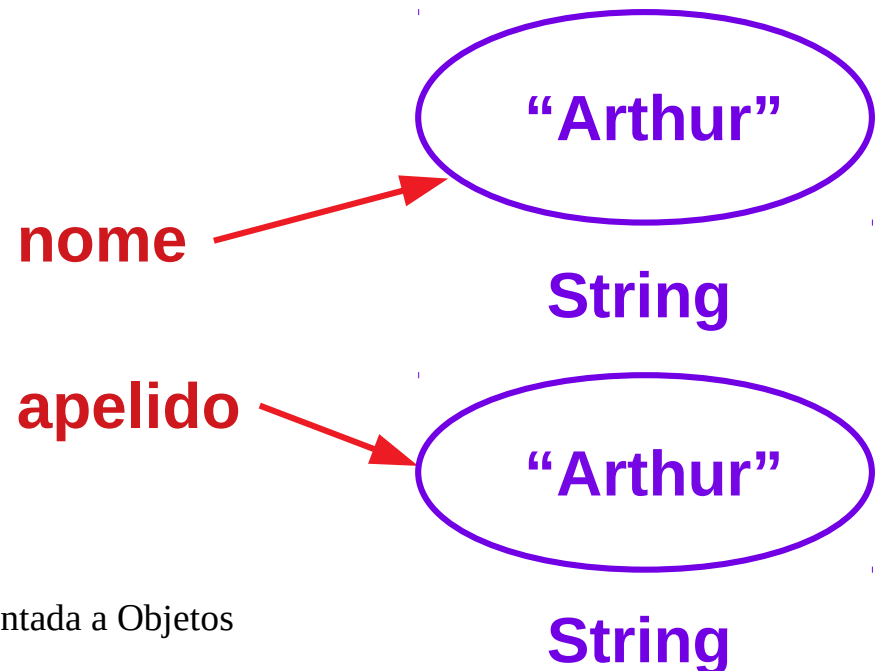
O teste continua sendo verdadeiro.
Java economiza memória e cria
um **objeto único**



Strings : Igualdade (mais um)

```
String nome = new String("Arthur");  
String apelido = "Arthur";  
  
if (nome == apelido) {  
    System.out.println("Nome e apelido são iguais");  
}
```

Agora o teste retornará falso, pois **cada variável aponta para um objeto diferente**. O conteúdo do objeto é irrelevante aqui.





Roteiro

- Utilização de Strings em Java
- Métodos principais da classe String
- Exercícios

Strings : método *equals()*

- A maneira correta de comparar se duas Strings tem o conteúdo igual é utilizando o método *equals()*

```
String nome1 = "Carlos";  
String nome2 = "Carla";
```

Aqui o resultado será **falso**

```
if (nome1.equals(nome2)) {  
    System.out.println("Os nomes são iguais!");  
} else {  
    System.out.println("Os nomes são diferentes!");  
}
```


Strings : método *equals()*

- A maneira correta de comparar se duas Strings tem o conteúdo igual é utilizando o método *equals()*

```
String nome1 = "Carlos";  
String nome2 = "Carlos";
```

Mas aqui o
resultado será **verdadeiro**

```
if (nome1.equals(nome2)) {  
    System.out.println("Os nomes são iguais!");  
} else {  
    System.out.println("Os nomes são diferentes!");  
}
```

equalsIgnoreCase()


- Esse método testa a igualdade do conteúdo sem se importar com letras maiúsculas e/ou minúsculas

```
String nome1 = "Carlos";  
String nome2 = "cArLoS";
```

```
if(nome1.equalsIgnoreCase(nome2)) {  
    System.out.println("Os nomes são iguais!");  
}
```

String : *compareTo()*

- Esse método pode retornar:
 - **0** se as strings forem **iguais**,
 - um número **negativo** se a string que invoca o compareTo for **menor** que a string que é passada como um argumento e;
 - um número **positivo** se a string que invoca o compareTo for **maior** que a string que é passada como argumento.



compareTo() : Exemplo

```
String nome1 = "Carlos";  
String nome2 = "Carla";  
String nome3 = "Luiza";
```

```
System.out.println("nome2.compareTo(nome1) = "+nome2.compareTo(nome1));  
System.out.println("nome1.compareTo(nome2) = "+nome1.compareTo(nome2));  
System.out.println("nome3.compareTo(nome2) = "+nome3.compareTo(nome2));
```

Resultados:

```
nome2.compareTo(nome1) = -14  
nome1.compareTo(nome2) = 14  
nome3.compareTo(nome2) = 9
```

Outros métodos úteis

```
String texto = "A API de Strings é uma das mais utilizadas na linguagem Java";
String linguagem = texto.substring(texto.indexOf("Java"), texto.length());

if (linguagem.compareToIgnoreCase("java") == 0) {
    System.out.println("compareToIgnoreCase: Encontrei a linguagem! Ela é "
        + linguagem);
}

if (linguagem.compareTo("java") == 0) {
    System.out.println("compareTo: Encontrei a linguagem! Ela é "
        + linguagem);
}
```


Outros métodos úteis

- ***substring(início, fim)*** : Retorna uma parte do String (uma substring) começando pela posição indicada por *início* até a posição marcada por *fim*
- ***indexOf("texto")*** : Retorna o índice (posição) onde a palavra "texto" pode ser encontrada dentro da String. Se não encontrar, retorna -1
- ***length()*** : Retorna o comprimento em caracteres da String
- ***compareToIgnoreCase()*** : Compara duas Strings sem se importar com maiúsculas e minúsculas

Concatenação de Strings

- Duas Strings podem ser “juntadas” de duas formas diferentes:
 - Com o operador de adição : **+**
 - Com o método ***concat()***

```
String nomeCompleto = nome + sobrenome;
```

```
String nomeCompleto = nome.concat(sobrenome);
```



valueOf()

- Converte outros tipos de dados para o tipo String:

```
double numero = 102939939.939;  
boolean booleano = true;
```

```
System.out.println("Retorna Valor : " + String.valueOf(numero));  
System.out.println("Retorna Valor: " + String.valueOf(booleano));
```



charAt()

- Retorna o caractere em uma localização específica em uma String.
- Esse método possui um parâmetro do tipo inteiro que é usado como índice, retornando a partir dessa posição inserida nesse parâmetro.
- É importante lembrar que o índice sempre começa a ser contado do número 0 (zero) em diante.



charAt()

```
String nomeCurso = "JAVA";  
  
if(nomeCurso.charAt(1) == 'A') {  
    System.out.println("O caractere A está na posição 1");  
}
```




getChars()

- **Extrai os caracteres de um String, copiando-os para um vetor de caracteres.**
- Esse método possui os seguintes parâmetros de entrada:
 - ***srcBegin*** – Índice do primeiro caractere da string a ser copiada.
 - ***srcEnd*** - Índice depois da última string a ser copiada.
 - ***dst*** – O destino do array.
 - ***dstBegin*** – o início do deslocamento no array de destino.



getChars() : Exemplo

```
String nomeCurso = "Curso Java Web";  
  
//É A DIFERENÇA DO 1º E 2º PARÂMETRO DO MÉTODO getChars  
//SE DIMINUIR OS 2 O RESULTADO TEM QUE SER O MESMO INICIADO NO ARRAY  
  
char[] numIndice = new char[7];  
  
nomeCurso.getChars(2, 9, numIndice, 0);  
System.out.print("Valores Copiados \n");  
  
for(char caracter : numIndice) {  
    System.out.print "["+caracter+"]";  
}
```



startsWith()** e **endsWith()

- Os métodos ***startsWith*** e ***endsWith*** aceitam uma String e um número inteiro como argumentos, retornando um valor booleano que indica se a string inicia ou termina, respectivamente, com o texto informado a partir da posição dada.

startsWith() e *endsWith()*

- Exemplo:

```
String[] nomes = {"iniciado", "iniciando", "finalizado", "finalizando"};
```

```
for (String recebeNomes : nomes) {  
    if (recebeNomes.startsWith("in"))  
        System.out.printf("\"%s\" inicia com \"in\" \n", recebeNomes);  
}
```

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

```
for (String recebeNomes : nomes) {  
    if (recebeNomes.startsWith("ici", 2))  
        System.out.printf("\"%s\" inicia com \"ici\" na posição 2 \n", recebeNomes);  
}
```

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

```
for (String recebeNomes : nomes) {  
    if (recebeNomes.endsWith("ado"))  
        System.out.printf("\"%s\" termina com \"ado\" \n", recebeNomes);  
}
```



indexOf()

- Localiza a primeira ocorrência de um caractere em uma string. Se o método localizar o caractere, é retornado o índice do caractere na String, caso contrário retorna -1.
- Existem **duas versões** do *indexOf* que procuram caracteres em uma String.
 - 1ª versão – aceita um inteiro que é conhecido como o número do índice na String.
 - 2ª versão – aceita dois argumentos inteiros – o caractere e o índice inicial em que a pesquisa da String deve iniciar.



indexOf()

```
String letras = "abcaefghijklmcpqrsdeftuvz";
//TESTA indexOf PARA LOCALIZAR UM CARACTERE EM UM STRING
System.out.printf("Último 'c' está localizado no index %d\n", letras.indexOf('c'));
System.out.printf("Último 'a' está localizado no index %d \n", letras.indexOf('a', 1));

// -1 NÃO EXISTE
System.out.printf("'$' está localizado no index %d\n\n", letras.indexOf('$'));

//TESTA indexOf PARA LOCALIZAR UMA SUBSTRING EM UMA STRING
System.out.printf("\"def\" está localizado no index %d\n", letras.indexOf("def"));

//2 argumento string e outro o ponto inicial que começará a pesquisa
System.out.printf("\"def\" está localizado no index %d\n", letras.indexOf("def", 7));
System.out.printf("\"hello\" está localizado no index %d\n\n", letras.indexOf("hello"));
```



lastIndexOf()

- Localiza a última ocorrência de um caractere em uma string. O método procura do fim da string em direção ao começo, se encontrar o caractere é retornado o seu índice na string, caso contrário retorna -1.
- Existem **duas versões do *lastIndexOf*** que pesquisam por caracteres em uma string.
 - 1ª versão – utiliza um inteiro do caractere.
 - 2ª versão – aceita 2 argumentos – um inteiro de caractere e o índice a partir do qual iniciar a pesquisa de trás para frente.



lastIndexOf()

```
String letras = "abcdefghijklmnopqrstuvwxyz";  
  
// TESTA lastIndexOf PARA LOCALIZAR UM CARACTERE EM UMA STRING  
  
System.out.printf("Último 'c' está localizado no index %d\n",  
    letras.lastIndexOf('c'));  
System.out.printf("Último 'a' está localizado no index %d\n",  
    letras.lastIndexOf('a', 5));  
System.out.printf("Último '$' está localizado no index %d\n",  
    letras.lastIndexOf('$'));
```



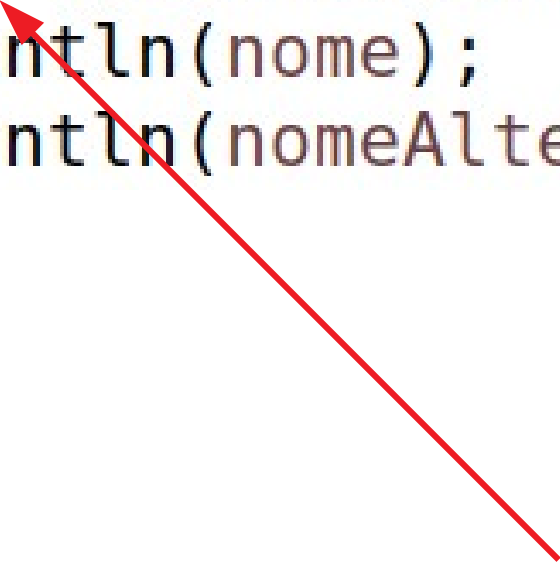
replace()

- Retorna um novo objeto contendo a String original com um trecho especificado substituído por outra expressão indicada.
- Esse método deixa a String original inalterada.
- A versão sobrecarregada do método ***replace*** permite substituir substrings em vez de caracteres individuais.




replace()

```
String nome = "mesquita";  
String nomeAlterado = nome.replace('e', 'o');  
nomeAlterado = nomeAlterado.replace('a', 'o');  
System.out.println(nome);  
System.out.println(nomeAlterado);
```



Notem que para alterarmos o valor da String original temos que fazer a **atribuição** da classe para ela mesma



uppercase() e *lowerCase()*

- *toUpperCase()*

- Retorna uma nova string com o conteúdo da original convertido para letras **maiúsculas**, mantendo a original inalterada.

- *toLowerCase()*

- De forma semelhante ao anterior, o *toLowerCase* retorna uma cópia de uma string com todas as letras convertidas para **minúsculo**, mantendo a original inalterada.

uppercase() e *toLowerCase()*

```
String nomeA = "joaquina";  
String nomeB = "Paulo";
```

```
System.out.println(nomeA.toUpperCase());  
System.out.println(nomeB.toLowerCase());
```



trim()

- Gera um novo objeto string, **removendo todos os caracteres de espaço em branco** que aparecem no início ou no fim da string em que o método opera.
- O método retorna um novo objeto string contendo a string sem espaço em branco inicial ou final.
- A string original permanece inalterada.



trim()

```
String s3 = "  espaços  ";
```

```
// MÉTODO TRIM - REMOVE OS ESPAÇOS
```

```
System.out.printf("s3 depois do trim = \"%s\"\n\n", s3.trim());
```



Dúvidas?





Atividades

- Execute as atividades presentes no documento

06.Lista.de.Exercícios.POO.pdf

Próximos passos



- Pacotes e Interfaces