



Application Delivery Fundamentals: Java

Java Excellence

accenture > **technology**

Module 5: Classe String

[illegible]

“CLASSE STRING”

- O modo mais direto de criar uma string é:

Variáveis do tipo String guardam referências a objetos, e não um valor, como acontece com os tipos primitivos;

- A classe String possui 15 métodos construtores, que nos permitem fornecer o valor inicial da string usando diferentes fontes, tais como arrays de caracteres: .

```
String str = new String("ACCENTURE");
```

Copyright © 2025 Accenture All Rights Reserved.

```
String str = "Técnicas de Programação II";  
  
int tamanho = str.length();  
  
System.out.println("Tamanho: " + tamanho);
```

```
String str = "Técnicas de Programação II";

System.out.println("Primeira Letra: " + str.charAt(0));

System.out.println("Ultima Letra: " + str.charAt(str.length()-1));
```

```
Primeira Letra: T
Ultima Letra: I
```

Classe String

As variáveis de objetos em Java fazem referencia aos objetos instanciados em memória.

CODE:

MEMORY:

[illegible]

- Lista completa:
<https://docs.oracle.com/javase/7/docs/api/java/lang/String.html>

public class String

String(String s)	<i>create a string with the same value as s</i>
String(char[] a)	<i>create a string that represents the same sequence of characters as in a[]</i>
int length()	<i>number of characters</i>
char charAt(int i)	<i>the character at index i</i>
String substring(int i, int j)	<i>characters at indices i through (j-1)</i>
boolean contains(String substring)	<i>does this string contain substring?</i>
boolean startsWith(String prefix)	<i>does this string start with prefix?</i>
boolean endsWith(String postfix)	<i>does this string end with postfix?</i>
int indexOf(String pattern)	<i>index of first occurrence of pattern</i>
int indexOf(String pattern, int i)	<i>index of first occurrence of pattern after i</i>
String concat(String t)	<i>this string, with t appended</i>
int compareTo(String t)	<i>string comparison</i>
String toLowerCase()	<i>this string, with lowercase letters</i>
String toUpperCase()	<i>this string, with uppercase letters</i>
String replace(String a, String b)	<i>this string, with as replaced by bs</i>
String trim()	<i>this string, with leading and trailing whitespace removed</i>
boolean matches(String regexp)	<i>is this string matched by the regular expression?</i>
String[] split(String delimiter)	<i>strings between occurrences of delimiter</i>
boolean equals(Object t)	<i>is this string's value the same as t's?</i>
int hashCode()	<i>an integer hash code</i>

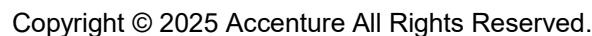
2f48696768207065726

65642c2f48696768207

Comparação

- Operador `==` compara tipos primitivos
- O método `equals` compara objetos

```
String s = new String("abc");  
String s2 = new String("abc");
```





Não é igual!

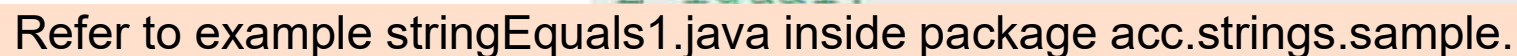


Retorna **true** se, e somente se, a string representa a mesma sequência de caracteres que o objeto passado como argumento.

```
String s = new String("abc");
String s2 = new String("abc");

if (s1.equals(s2))
    System.out.println("É igual!");
else
    System.out.println("Não é igual!");
```

É igual!



- **int compareTo(String s)**– compara duas strings lexicograficamente. Retorna um inteiro que indica se a string maior (retorno > 0), igual (retorno $= 0$) ou menor (retorno < 0) que a string passada como argumento.



Refer to example `StringCompareTo1.java`, `StringEndsWith` inside package `acc.strings.sample`.

Uma vez criado um objeto da classe String, ele não pode ser modificado.

- ```
String s = "abc";
String s2 = s;
s = s.concat("def");

System.out.println(s);
System.out.println(s2);
```

```
abcdef
abc
```

# Classe String - Imutabilidade

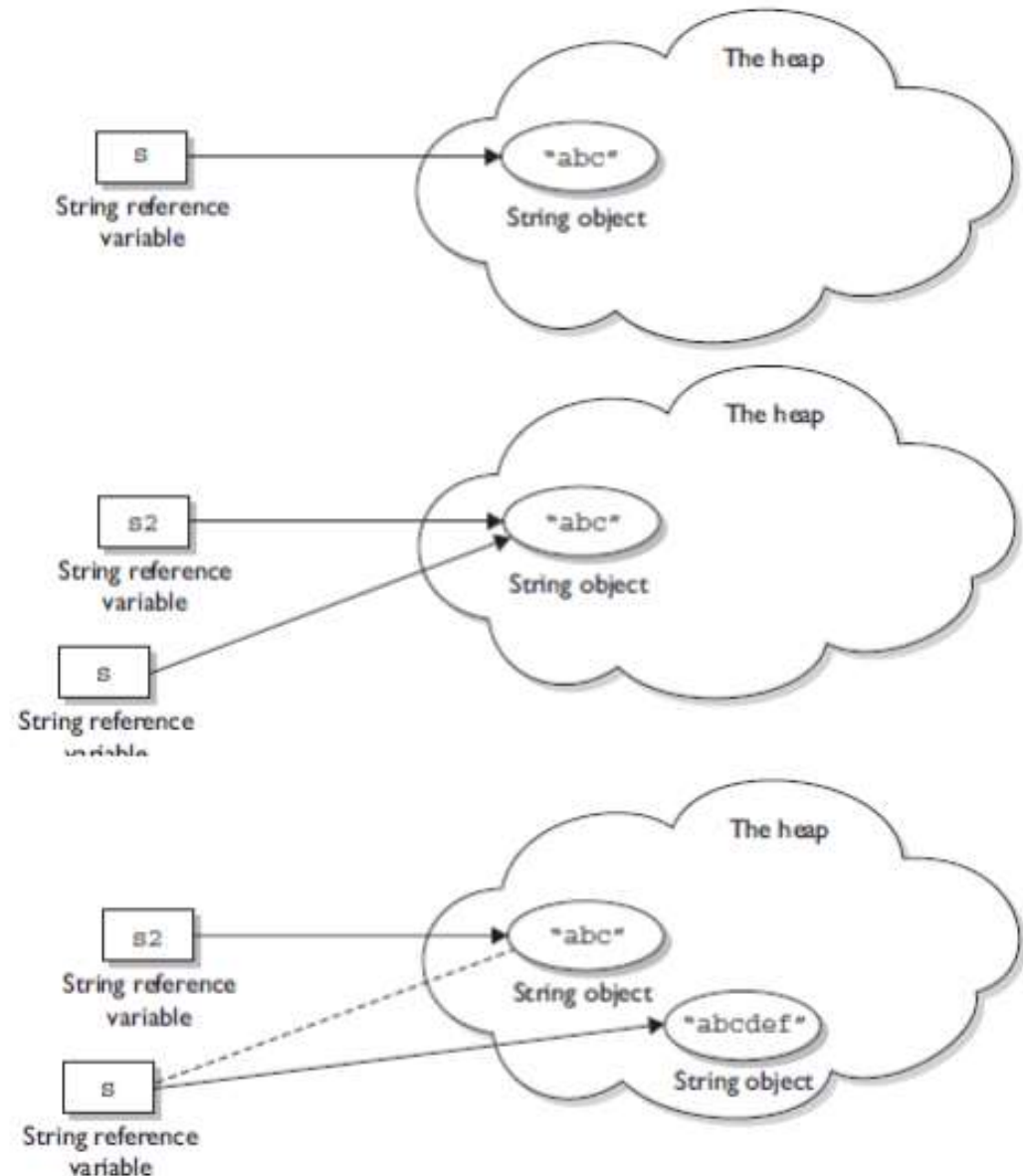
2f48696768207065726

65642c2f48696768207

```
String s = "abc";
```

```
String s2 = s;
```

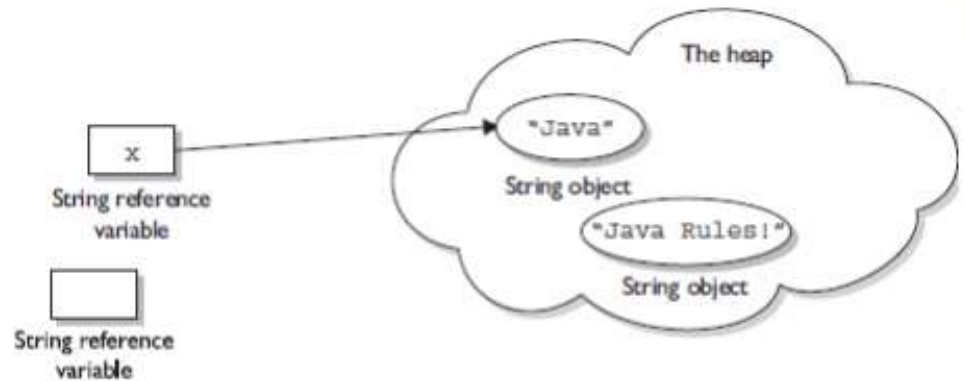
```
s = s.concat("def");
```



Refer to example `stringConcat1.java` inside package `acc.strings.sample`.

2f4869676820706572666f72d6d16e36352e2044656c6976657265642e2f4869676820706572666f72d6d16e36352e2044656c6976657265642e2f4869676820706572666f72d6d16e36352e2044656c6976657265642e2f4869676820706572666f72d6d16e36352e2044656c6976657265642e2f4869676820706572666f72d6d16e36352e2044656c6976657265642e2f48696768207

## Java



Copyright © 2025 Accenture All Rights Reserved.


# Classe String – StringBuffer e StringBuilder

- A classe **String** é **imutável**. Sendo assim, quando algo é concatenado a uma String, esta não é alterada. O que ocorre é a criação de uma nova String.
- Por isso trabalhar com uma mesma String diversas vezes pode ter um efeito colateral: gerar inúmeras Strings temporárias.
- Isto prejudica a performance da aplicação consideravelmente.

# Classe String – StringBuffer e StringBuilder

- Se for necessário trabalhar muito com a manipulação de uma mesma String (por exemplo, dentro de um laço), o ideal é utilizar a classe `StringBuffer`.
- A classe `StringBuffer` representa uma sequência de caracteres.
- Diferentemente da `String`, ela é mutável e, por isso, evita problemas de performance.

A classe **StringBuffer** é *sincronizada* para acesso concorrente, enquanto que a **StringBuilder** não tem *sincronização*.

-  Refer to example StringBuffer1.java StringBuilder.java inside package acc.strings.sample.


# Classe String – StringBuffer e StringBuilder

## Use **String** para manipular com valores constantes:

- Use **StringBuffer** para alterar textos: – Acrescentar, concatenar, inserir, etc.
- Prefira usar **StringBuffer** para construir Strings
  - Concatenação de strings usando "+" é extremamente cara.



- **String nextLine()**– Retorna a ultima linha de texto digitada no console;

 Refer to example `StringScanner.java` inside package `acc.strings.sample`.



É possível analisar os caracteres de uma Strings usando o método **charAt**. Exemplo:

```
public static int conta_letra(String str, char letra)
{
 int i, total = 0;
 for (i = 0; i < str.length(); i++)
 {
 if (str.charAt(i) == letra)
 total++;
 }
 return total;
}
```

```
String str1 = "Tecnicas de Programacao II";
System.out.println("Total: " + conta letra(str1, 'a'));
```

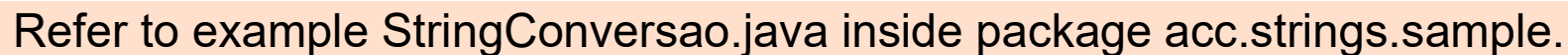


Refer to example `StringContaLetras.java` inside package `acc.strings.sample`.

- ```
String str1 = "85.1";  
double num1 = Double.parseDouble(str1);
```



[illegible]



Classe String

toString

- O método `toString()` faz parte de todos os objetos em java por ser um método da classe `Object`
- Sobrescrevendo o comportamento deste método podemos definir uma representação de um objeto como uma `String`

[illegible]

toString

```
class Ponto {
    int x, y;
    public String toString() {
        return "Ponto(" + x + ", " + y + ")";
    }
}
```



Refer to example `StringToString.java` inside package `acc.strings.sample`.

-

