

Module 5: Classe String

•

"CLASSE STRING"

Strings são sequências de caracteres.

Na linguagem Java, strings são instâncias da classe String.

• O modo mais direto de criar uma string é:

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

• Sempre que o compilador encontra uma string no código de um programa ele cria um objeto da classe.

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

Como qualquer outro objeto, uma string pode ser criada pelo operador **new**.

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

```
Char data[] = { 'A', 'C', 'C', 'E', 'N', 'T', 'U', 'R', 'E'}
String str = new String(data);

String str = new String("ACCENTURE");

String str = "ACCENTURE";
```

- Classe String fornece um conjunto de métodos para a manipulação de cadeias de caracteres.
- Um destes métodos é o método length, o qual retorna um valor inteiro igual ao número de caracteres contidos na string:

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

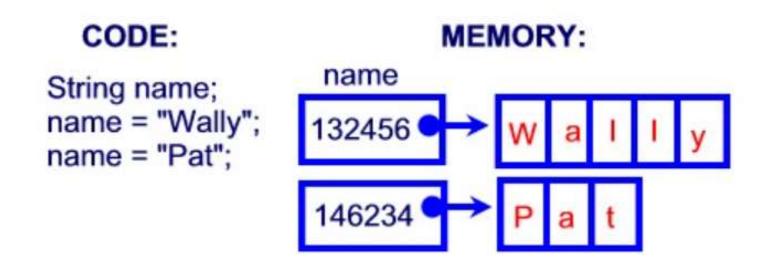
Outro método da classe String é o método *charAt*, o qual permite o acesso individual aos caracteres contidos na string:

```
String str = "Técnicas de Programação II";
System.out.println("Primeira Letra: " + str.charAt(0));
System.out.println("Ultima Letra: " + str.charAt(str.length()-1));
```

Resultado:

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

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



• A classe String possui um método **estático**, chamado **format**, que nos permite formatar uma string.

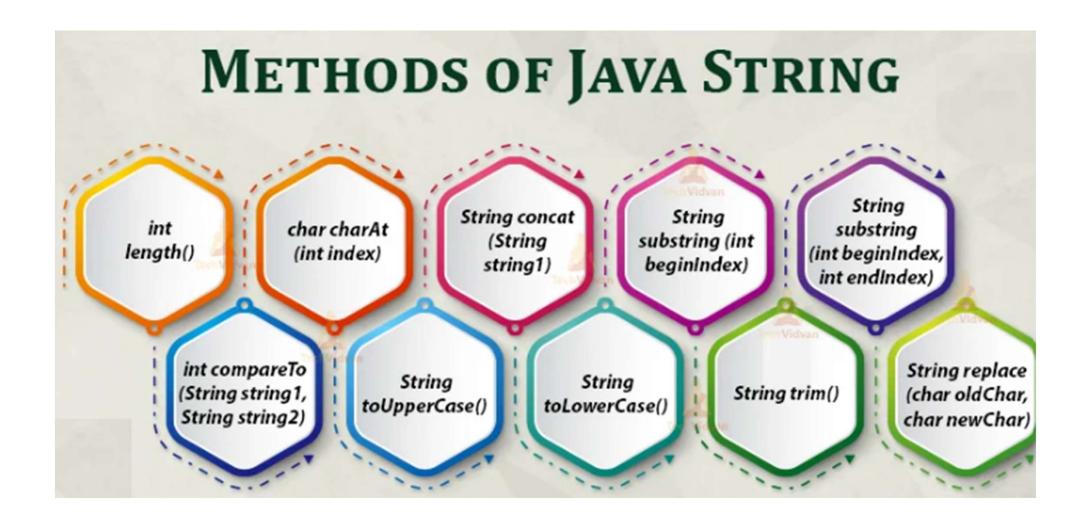
```
String str1;
str1 = String.format("Alunos: %d \nMédia: %.2f", 25, 7.45);
System.out.println(str1);
```

Resultado:

Alunos: 25

Média: 7,45

Métodos String



- Alguns outros métodos da classe String:
 - boolean contains(String s)
 - int indexOf(String str)
 - String replace(char oldChar, char newChar)
 - String substring(int beginIndex, int endIndex)
 - String toLowerCase()
 - String toUpperCase()
 - String trim() .

Lista completa:

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

2f48696768207065726

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

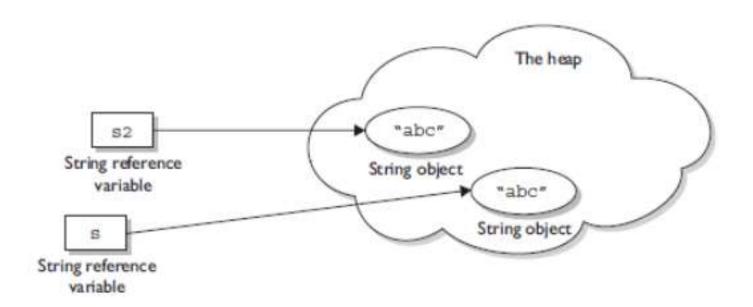
65642c2f48696768207

Comparação

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

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

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

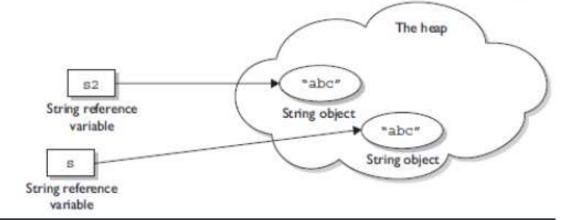


Ao usarmos o operador == para compararmos variáveis de objetos, estamos verificando se ambas apontam para o mesmo objeto instanciado em memória.

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

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

Resultado: Não é igual!



Para compararmos o conteúdo de uma String devemos utilizar os métodos de comparação fornecidos pela classe String:

boolean equals(Object anObject)

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

como argumento.

Exemplo:

```
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!");
```

Resultado:

É iqual!

R

Refer to example stringEquals1.java inside package acc.strings.sample.

A classe String também fornece outros métodos para comparação de Strings:

- boolean endsWith(String suffix)— retorna true se a string termina com a string passada como argumento.
- boolean startsWith(String prefix)
 – retorna true se a string começa com a string passada como argumento.
- 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.

Em Java, strings são imutável.

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

 A classe String possui vários métodos que parecem alterar o conteúdo de uma string, mas estes métodos na verdade fazem é criar e retornar uma nova string que contém o resultado da operação.

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

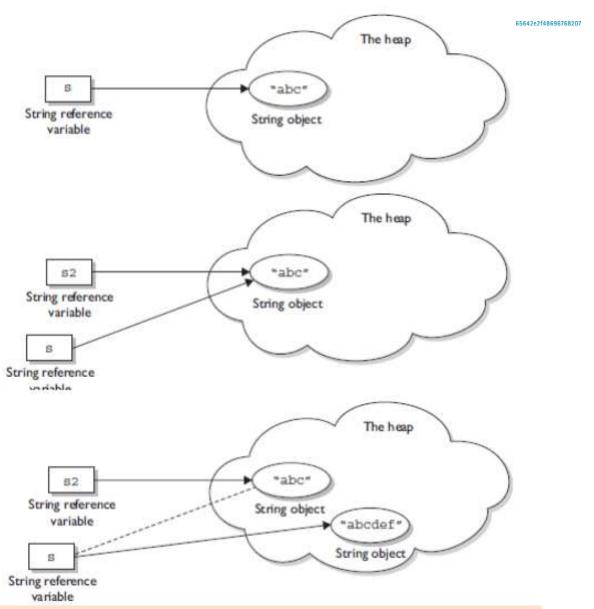
System.out.println(s);
System.out.println(s2);
```

```
Resultado:
abcdef
abc
```

Classe String - Imutabilidade

2f48696768207065726

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



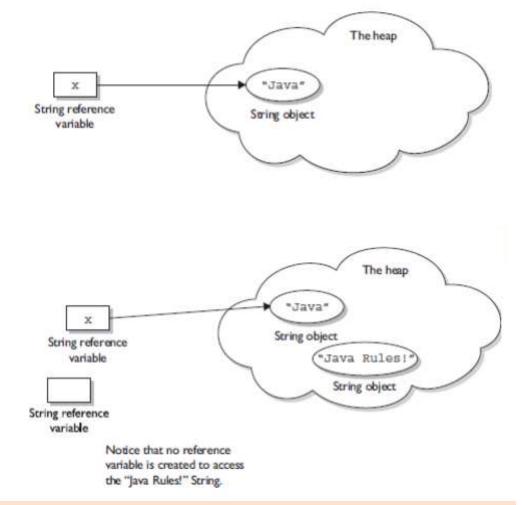


Refer to example stringConcat1.java inside package acc.strings.sample.

```
String x = "Java";
x.concat(" Rules!");
System.out.println(x);
```

Resultado:

Java





Refer to example stringConcat2.java inside package acc.strings.sample.

- 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.

- 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.

Ambas as classes possuem a maioria dos métodos da classe String, além de métodos de alteração.

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

- Os principais métodos destas classes são:
- append(...)
 – adiciona o parâmetro ao final da string;
- insert(int offset, ...) insere o parâmetro na posição;
- setCharAt(int index, char ch)- altera o caractere na posição;



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

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

- Valores literais;
- Textos carregados de fora da aplicação;
- Textos em geral que não serão modificados intensivamente.
- Use **StringBuffer** para alterar textos: Acrescentar, concatenar, inserir, etc.
- Prefira usar StringBuffer para construir Strings
 - Concatenação de strings usando "+" é extremamente cara.

Classe String – Leitura de strings

A classe **java.util.Scanner**, além de fornecer funções para a leitura de variáveis simples (int, float, double...), também fornece funções para leitura de Strings:

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

```
String nome;
Scanner entrada = new Scanner(System.in);
System.out.println("Digite o seu nome:");
nome = entrada.nextLine();
System.out.println(nome);
```

i

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

Classe String – Manipulando Caracteres

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;
}</pre>
```

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

i

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

Classe String – Conversão de Strings

Converter de String para int:

```
String str1 = "52";
int num1 = Integer.parseInt(str1);
```

Converter de String para float:

```
String str1 = "28.6";
float num1 = Float.parseFloat(str1);
```

Converter de String para double:

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

i

Refer to example StringConversao.java inside package acc.strings.sample.

Classe String – Conversão de Strings

O método **valueOf** é um método estático que recebe um argumento de qualquer tipo e o converte em um objeto String.

Classe String – Conversão de Strings

O método **valueOf** é um método estático que recebe um argumento de qualquer tipo e o converte em um objeto String.

Converter de int para String:

```
int num1 = 52;
String str1 = String.valueOf(num1);
```

Converter de float para String:

```
float num1 = 28.6f;
String str1 = String.valueOf(num1);
```

Converter de String para double:

```
double num1 = 85.1;
String str1 = String.valueOf(num1);
```

i

Refer to example StringConversao.java inside package acc.strings.sample.

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

toString

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



Refer to example StringtoString.java inside package acc.strings.sample.

toString

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

Questions and Comments

 What questions or comments do you have?



