JAVA

CLASSE **STRING**

CLASSE **STRING**

- Strings são utilizadas frequentemente em várias linguagens de programação, não apenas Java.
- Em JAVA uma **String** é uma sequência de caracteres, e não um Array de caracteres.
- O importante é entender que String não é um tipo de dado, mas sim uma **classe**. E suas variáveis são, na verdade, objetos dessa classe.
- Em síntese, tanto um Array de caracteres como uma String se apresentam da mesma forma.

CLASSE **STRING**

- Então, qual a vantagem de se usar uma classe e não um Array de tipo primitivo?
- A resposta se baseia exatamente no uso da orientação a objetos e no fato de que existem muitos métodos que podemos utilizar em um objeto instanciado da classe String, mesmo sendo os objetos desta classe imutáveis, ou seja, uma vez instanciados não podemos mudar o que está guardado dentro do objeto String.

CLASSE **STRING**
MÉTODOS

- **concat()**
- **length()**
- **toUpperCase()**
- **toLowerCase()**
- **equals()**
- **compareTo()**
- **startsWith()**
- **endsWith()**
- **charAt()**
- **indexOf()**
- **replace()**
- **contains()**
- **split()**
- **substring()**

CLASSE **STRING**

CONCAT()

- Concatenação nada mais é do que juntar Strings numa só. Isto pode ser feito de duas formas: uma usando o método **concat()** da classe String ou usando o sinal de adição (+) como operador de concatenação.

- O método concat() retorna uma nova string formada da junção da string principal com a string indicada como parâmetro.

- De uma forma mais simples, podemos usar o + para juntar várias strings ao mesmo tempo.

- Concatenção ▶

CONCATENAÇÃO

- Concatenação é o ato de unir duas ou mais cadeias de caracteres (strings).
- Por muito tempo, operações envolvendo strings eram os pesadelos de qualquer programador, pois havia a necessidade de tratar o elemento pelo seu dado bruto, ou seja, caracter por caracter. Então, juntar por exemplo, duas frases era um trabalho árduo.
- Com o surgimento da orientação a objeto e o advento do Java, as operações envolvendo strings foram muito simplificadas. A melhoria mais significativa nesse assunto, sem sombra de dúvidas, foi utilizar a concatenação de strings.

CONCATENAÇÃO

- A concatenação de strings é dada pelo operador +, mas não o confunda com o operador de adição que utiliza o mesmo símbolo.
- Dessa forma, com apenas este símbolo, podemos unir duas cadeias de caracteres diferentes em apenas uma.
- Veja este exemplo, que apesar de bobo, ilustra exatamente o que acontece na concatenação

João + zinho = Joãozinho;

Passa + tempo = Passatempo;

beija + - + flor = beija-flor.

CONCATENAÇÃO

- Exemplo:

```
public class ConcatenacaoSimples {
    public static void main(String[] args) {
        String palavra1 = "tele";
        String palavra2 = "fone";
        System.out.println(palavra1 + palavra2);
    }
}
```

Será impresso na Tela:

telefone

CONCATENAÇÃO

- Exemplo: Utilizando a função **concat**()

```
public class ConcatenacaoConcat {
    public static void main(String[] args) {
        String palavra1 = "tele";
        palavra1 = palavra1.concat(“fone”);
        System.out.println(palavra1);
    }
}
```

Será impresso na Tela:

telefone

CLASSE **STRING**

LENGTH()

- Método assessor que retorna o **tamanho** da String.
- Exemplo:

```
public class TamanhoString {
	public static void main(String[] args) {
		String str = "TI Expert";
		System.out.println(str + " possui " + str.length() + " caracteres.");
	}
}
```

Será impresso na Tela:

TI Expert possui 9 caracteres.

CLASSE **STRING**
TOUPPERCASE()

- Podemos facilmente deixar todas as letras de uma sequência de caracteres em **maiúscula** usando o métodos toUpperCase().

Exemplo:

```
public class StringMaiusculoMinusculo {
    public static void main(String[] args) {
        String hexadecimal = "#aa33ff";
        System.out.println(hexadecimal.toUpperCase());
    }
}
```

CLASSE **STRING**
TOLOWERCASE()

- Podemos facilmente deixar todas as letras de uma sequência de caracteres em **minúscula** usando o métodos toUpperCase().

Exemplo:

```
public class StringMaiusculoMinusculo {
    public static void main(String[] args) {
        String hexadecimal = "#aa33ff";
        System.out.println(hexadecimal.toLowerCase());
    }
}
```

CLASSE **STRING**

COMPARAÇÃO

- Há várias formas de se fazer comparações com uma string, mas vale sempre lembrar que Java é case sensitive - diferencia letras maiúscula de minúsculas e vice-versa. Embora haja métodos próprios para comparações em que não há importância em diferenciar letras maiúsculas e minúsculas.
- A função equals() é case sensitive, para fazer uma comparação ignorando esta característica basta usar o método equalsIgnoreCase().
- A comparação mais simples é usando o próprio operador de igualdade (==), mas por se tratar de um objeto é preferível que se use o método específico chamado equals().

CLASSE **STRING**

EQUALS()

Diferença entre "==" e "equals"

A princípio possuem o mesmo comportamento, mas o primeiro está presente somente em tipos de dados não primitivos, enquanto o operador de igualdade lógica pode ser utilizado em tipos primitivos e não primitivos.

O operador **lógico de igualdade**, representado por **==**, retorna true se as variáveis que estão sendo comparadas são iguais e false caso contrário. É para comparar o valor que uma variável guarda. É mais útil para variáveis de tipo primitivos.

equals um **método** para comparar o valor significativo de um objeto cuja variável do tipo referência aponta. É através deste método que são feitas as verificações de igualdade entre os **objetos**.

CLASSE **STRING**

EQUALS()

```
public class ComparacaoIgualdadeString {
	public static void main(String[] args) {
		String string1 = "TI Expert";
		String string2 = "ti expert";
		System.out.println("São iguais? (case sensitive)");
		System.out.println(string1.equals(string2) ? "sim" : "não");

		System.out.println("São iguais? (sem case sensitive)");
		System.out.println(string1.equalsIgnoreCase(string2) ? "sim" : "não");
}
}
```

CLASSE **STRING**

COMPARETO()

- Há também uma forma de comparar strings lexicograficamente.

- Dessa forma, podemos verificar se uma string é identica (quando retorna-se 0), ou se ela tem um valor menor (quando retorna-se um número abaixo de 0) ou se ela tem um valor maior (quando retorna-se um número acima de 0).

- O método para fazer tais comparações é o compareTo() (case sensitive) ou compareToIgnoreCase() (sem case sensitive).

CLASSE **STRING**

COMPARETO()

```
public class ComparacaoString {
	public static void main(String[] args) {
		String string1 = "TI Expert";
		String string2 = "ti expert";
		int comparacao = string1.compareTo(string2);
		System.out.println("Comparação entre string1 e string2 (sensitive case)");
		if (comparacao > 0) {
			System.out.println("string1 é lexicograficamente maior que string2");
		} else if (comparacao < 0) {
			System.out.println("string1 é lexicograficamente menor que string2");
		} else {
			System.out.println("string1 é lexicograficamente igual a string2");
		}
	}
}
```

CLASSE **STRING**
STARTSWITH() E ENDSWITH()

- Outra forma de fazer comparações é fazer testes no início e no fim de uma string.
- Podemos fazer tais comparações usando dois métodos: startsWith() e endsWith().
- StartsWith() verifica se há uma string no começo de outra string. StartsWith() também possui um segundo parâmetro opcional que determina a compensação inicial, ou seja, caso necessite verificar a string não da posição 0, mas de uma posição mais adiante.
- EndsWith() verifica se há uma string no final de outra string. Diferentemente de startsWith(), o método endsWith() não possui compensação.

CLASSE **STRING**

STARTSWITH()

```
public class InicioFim {
	public static void main(String[] args) {
		String string1 = "http://www.tiexpert.net";
		System.out.println("A string " + string1 + " é:");
		if (string1.startsWith("http:")) {
			System.out.println("uma URL");
		}
		if (string1.startsWith("www", 7)) {
			System.out.println("uma página da web");
		}
	}
}
```

CLASSE **STRING**

ENDSWITH()

```
public class InicioFim {
	public static void main(String[] args) {
		String string1 = "http://www.tiexpert.net";
		System.out.println("A string " + string1 + " é:");
		if (string1.endsWith(".br")) {
			System.out.println("um site registrado no Brasil");
		} else {
			System.out.println("não é um site registrado no Brasil");
		}
	}
}
```

CLASSE **STRING**

CHARAT()

- Podemos também obter um caracter que se encontra em alguma posição dentro da string. Para isso, usaremos o método CharAt().

- CharAt() recebe um inteiro como argumento que indica a posição que queremos da string.

- Importante: Strings seguem o mesmo conceito de vetores e arrays, portanto, seu primeiro caracter não está na posição 1, mas na posição 0.

CLASSE **STRING**

CHARAT()

```
public class ExtrairCaracter {
	public static void main(String[] args) {
		String string1 = "TI Expert";
		char caracter = string1.charAt(3);
		System.out.println("O 4o. caracter desta string é " + caracter);
	}
}
```

CLASSE **STRING**

INDEXOF()

- A classe String possui uma forma de encontrar o índice da primeira ocorrência de uma string dentro de outra string.
- O método indexOf() retorna um número inteiro que indica exatamente em que posição ocorre uma string de busca, ou retorna um valor menor que 0 caso não encontre o valor requisitado. Assim como startsWith(), indexOf() também possui um segundo argumento que determina a compensação a ser feita caso a busca não possa ser feita desde a posição 0, mas de uma posição posterior.
- Além do método indexOf(), há também o método lastIndexOf() que faz a mesma coisa que indexOf(), porém de forma recursiva (de trás para frente ou do fim da string para o começo)

CLASSE **STRING**

INDEXOF()

```
public class IndiceString {
	public static void main(String[] args) {
		String string1 = "www.tiexpert.net";
		int posicao;
		posicao = string1.indexOf("tiexpert");
		if (posicao >= 0) {
			System.out.println("A string tiexpert começa na posição " + posicao);
		} else {
			System.out.println("Não há tiexpert na string");
		}
		posicao = string1.lastIndexOf(".com");
		if (posicao >= 0) {
			System.out.println("A string .com começa na posição " + posicao);
		} else {
			System.out.println("Não há .com na string");
		}
	}
}
```

CLASSE **STRING**

REPLACE()

- Podemos substituir todas as ocorrências de uma string por uma nova string resultando em uma nova string de retorno.

- Para executarmos esta operação usamos o método replace() que tem dois parâmetros: o primeiro será o que desejamos procurar dentro da string e o segundo a string que colocaremos no lugar da antiga.

- Nota: replace também funciona com o tipo primitivo char.

CLASSE **STRING**

REPLACE()

Exemplo:

```
public class SubstituirString {
	public static void main(String[] args) {
		String string1 = "http://tiexpert.net";
		System.out.println(string1.replace("http://", "www."));
	}
}
```

CLASSE **STRING**

CONTAINS()

- Um método muito útil para verificar o conteúdo de uma string é o contains().
- Contains() retorna verdadeiro (true) se houver a sequência de caracteres especificada no parâmetro.

```
public class VerificarConteudo {
    public static void main(String[] args) {
        String string1 = "http://www.tiexpert.net";
        System.out.println("É uma página da web?");
        System.out.println(string1.contains("www.") ? "sim" : "não");
    }
}
```

CLASSE **STRING**

SPLIT()

- Também é possível dividir uma string em um vetor ou array de strings, o que possibilita trabalhar com pedaços menores e mais lógicos da string.
- Para dividir a string podemos usar o método split(). O método split usa uma expressão regular para fazer a divisão.
- Para simplificar o entendimento, podemos usar como parâmetro uma outra string como base. Mas vale atentar no fato de que a expressão utilizada para ser feita a quebra desaparece no fim do processo.
- Ex.: Se minha string for 15:30:00, se utilizarmos os dois pontos (:) para a separação conseguiremos um array com 3 posições que seriam: [0]->15, [1]->30, [2]->00.

CLASSE **STRING**

SPLIT()

```
public class Exemplo {
public static void main(String[] args) {
String string1 = "15:30:00";
String[] stringDividida = string1.split(":");
for (int i = 0; i < stringDividida.length; i++){
System.out.println("stringDividida[" + i + "]=" + stringDividida[i]);
        }
    }
}
```

CLASSE **STRING**

SUBSTRING()

Substring é uma porção ou parte da string principal da qual pode formar outra string, ou seja, uma substring é uma string formada a partir de uma string principal.

Para obtermos uma substring usamos um método homônimo (de mesmo nome) com um ou dois parâmetros.

O primeiro parâmetro, obrigatório, é a posição que a substring deve iniciar (lembrando-se que strings sempre começam da posição 0). O segundo parâmetro, opcional, é a posição final da substring, caso este parâmetro não seja indicado, a substring sempre irá até o último caracter encontrado.

CLASSE **STRING**

SUBSTRING()

Nota: O caracter na posição inicial fará parte da substring, porém o caracter na posição final não faz parte da substring. Vejamos o exemplo: Se tivermos uma string de 9 caracteres, portanto, de 0 a 8 posições, se fizermos a substring da seguinte forma: stringOriginal.substring(3,8);

0	1	2	3	4	5	6	7	8
T	I		E	X	P	E	R	T

O resultado será Exper. Então, se quisermos que a 8ª posição entre, devemos usar a posição seguinte 9, assim: stringOriginal.substring(3,9);

CLASSE **STRING**

SUBSTRING()

```
public class ExemploSubstring {
	public static void main(String[] args) {
		String url = "www.tiexpert.net";
		String dominio = url.substring(4);
		String site = url.substring(url.indexOf('.') + 1, url.lastIndexOf('.'));
		System.out.println("Análise da string:");
		System.out.println("Domínio: " + dominio);
		System.out.println("Site: " + site);
	}
}
```