

LINGUAGEM DE PROGRAMAÇÃO ORIENTADA À OBJETOS – LPOO

Aula 09 – String, StringBuilder e StringBuffer



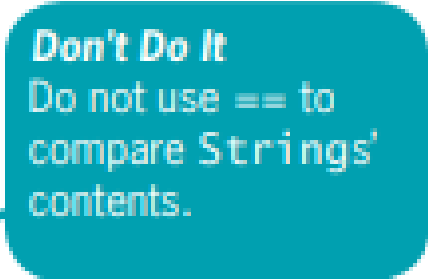
TÓPICOS DA AULA

- Identificar problemas em manipular Strings
- Declarar e usar objetos String
- Conhecer e usar os métodos da classe String
- Conhecer e usar os métodos da classe StringBuilder e StringBuffer

PROBLEMAS DURANTE A MANIPULAÇÃO DE STRINGS

- Manipular Strings fornece alguns desafios para o início do Java programador. Vamos analisar a classe **TryToCompareStrings** abaixo.

```
import java.util.Scanner;
public class TryToCompareStrings
{
    public static void main(String[] args)
    {
        String aName = "Carmen";
        String anotherName;
        Scanner input = new Scanner(System.in);
        System.out.print("Enter your name > ");
        anotherName = input.nextLine();
        if(aName == anotherName)
            System.out.println(aName + " equals " + anotherName);
        else
            System.out.println(aName + " does not equal " + anotherName);
    }
}
```



Don't Do It
Do not use == to compare Strings' content.

PROBLEMAS DURANTE A MANIPULAÇÃO DE STRINGS

- A classe sempre vai produzir resultados incorretos. O problema decorre de o fato de que em Java, **String é uma classe e cada String criada é um objeto.**
- Como um objeto, **uma variável do tipo String não é um tipo de dados simples - é uma referência**; ou seja, uma variável que contém um endereço de memória.
- Portanto, quando você compara dois objetos String usando o **operador ==**, **você não está comparando seus valores, mas seu endereço de memória.**

PROBLEMAS DURANTE A MANIPULAÇÃO DE STRINGS

- Se você declarar dois objetos String e inicializar ambos com o mesmo valor, o valor é armazenado apenas uma vez na memória e as duas referências de objeto **mantêm o mesmo endereço de memória.**
- Considere o seguinte exemplo em que o mesmo valor é atribuído a duas Strings. **O motivo da saída no exemplo a seguir é enganoso.** Quando você escreve o seguinte código, a saída é **Strings are the same**

```
String firstString = "abc";  
String secondString = "abc";  
if(firstString == secondString)  
    System.out.println("Strings are the same");
```

PROBLEMAS DURANTE A MANIPULAÇÃO DE STRINGS

- Felizmente, a classe String fornece uma série de métodos úteis que comparam Strings da maneira que você normalmente pretende.
- O métodos **equals()** e **equalsIgnoreCase()** da classe String avalia **o conteúdo** de dois objetos String para determinar se eles são **equivalentes**. O método retorna verdadeiro se os objetos têm conteúdos **idênticos**.

```
import java.util.Scanner;
public class CompareStrings
{
    public static void main(String[] args)
    {
        String aName = "Carmen";
        String anotherName;
        Scanner input = new Scanner(System.in);
        System.out.print("Enter your name > ");
        anotherName = input.nextLine();
        if(aName.equals(anotherName))
            System.out.println(aName + " equals " + anotherName);
        else
            System.out.println(aName + " does not equal " + anotherName);
    }
}
```

PROBLEMAS DURANTE A MANIPULAÇÃO DE STRINGS

- Em Java, o valor de uma String é fixado depois que a String é criada; **Strings são imutável.**

Veja o código a seguir:

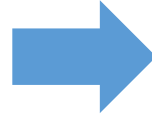
```
String userName = "Danilo";  
userName.toUpperCase();  
System.out.println(userName)
```

Você não alterou o conteúdo da variável userName, na instrução `userName.toUpperCase()`. Em vez disso, você armazenou “DANILO” em um novo local da memória do computador e armazenou o novo endereço de memória na variável `userName`.

PROBLEMAS DURANTE A MANIPULAÇÃO DE STRINGS

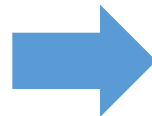
Para alterar corretamente o valor do conteúdo, demos **ATRIBUIR** o resultado da operação na variável.

```
String userName = "Danilo";  
userName.toUpperCase();  
System.out.println(userName)
```



```
String userName = "Danilo";  
userName = userName.toUpperCase();  
System.out.println(userName)
```

```
String userName = "Danilo";  
userName.replace('a', '@');  
System.out.println(userName);
```



```
String userName = "Danilo";  
userName = userName.replace('a', '@');  
System.out.println(userName);
```


DECLARANDO E USANDO A CLASSE STRING

- A classe String é definida em **java.lang.String**, que é **importado automaticamente** para cada programa que você escreve.
- Para declarar uma variável do tipo String, precisamos instanciar a classe String, usando a palavra reservada **new**

```
String nomeVariavel = new String ();
```

- Para iniciar a variável com um valor inicial, podemos fazer da seguinte maneira

```
String userName = new String ("Danilo Pereira");
```

EMPTY AND NULL

- Os programadores costumam ficar confusos com a diferença entre Strings **vazias** e Strings **nulas**. Vamos analisar as seguintes declarações de 3 variáveis do tipo String

```
String userName1 = "";  
String userName2 = null;  
String userName3;
```

- A String **userName1** faz **referência a um endereço de memória onde nenhum caractere é armazenado.**
- A String **userName2** usa a palavra-reservada em Java **null** para que indicar que a variável ainda não tem um endereço de memória.
- A String **userName3** **também é uma String nula por padrão**

CONVERTENDO STRING EM NÚMEROS

- Se uma String contém **todos os números**, como em "649", você pode convertê-la de uma String em um número então você **pode usá-lo para aritmética ou como qualquer outro número**.
- Por exemplo, suponha você pede a um usuário para inserir um salário em uma caixa de diálogo de entrada. **Usando o método JOptionPane.showInputDialog(), o valor retornado é sempre uma String**.
- Para ser capaz de usar o valor em **operações aritméticas**, você deve converter a String em um número.

CONVERTENDO STRING PARA NÚMEROS

- Para converter uma String em um inteiro, você usa a classe Integer. . Por exemplo, a instrução a armazena o valor do tipo String "649", variável inteira anInt:

```
int intValue = Integer.parseInt("649");
```

- Outros exemplos:

```
double doubleValue = Double.parseDouble("147.82")
```

*Além de Double e Integer, outras classes de wrapper, como Float e Long também fornecem métodos como **parseFloat()** e **parseLong()**.*

ALGUNS METODOS DA CLASSE STRING

- **length()** -> Retorna a quantidade de caracteres da String
- **trim()** -> Remove os espaços da esquerda e a direita
- **replace(String regex, String new)** -> Troca o conteúdo da String pelo conteúdo do parâmetro.
- **split(String regex)** -> Divide a String de acordo com parâmetro regex
- **charAt(int index)** -> Devolve o caracter que está na posição index.
- **concat(String str)** -> Faz a junção (no final) da String original com a string passada como parametro.

ALGUNS METODOS DA CLASSE STRING

- **startsWith(String prefix)** -> Retorna true se a String **inicia** com o valor passado como parâmetro (case sensitive – cuidado)
- **endsWith(String prefix)** -> Retorna true se a String **termina** com o valor passado como parâmetro (case sensitive – cuidado)
- **toUpperCase(String str)** -> Converte a String para maiúscula.
- **toLowerCase(String str)** -> Converte a String para minúscula.
- **substring(int beginIndex, int endIndex)** -> Retorna a substring de acordo com o índices iniciais e finais informados.
- **isBlank()** -> Retorna true se a String é vazia ou contém apenas espaços em brancos.

STRING BUFFER x STRING BUILDER

StringBuffer -> Uma *thread-safe* — isto é, que manipula EDs (estrutura de dados) compartilhadas entre diversas threads de forma segura —, sequência de caracteres mutáveis. Uma StringBuffer é como uma String, mas pode ser modificada, pois o tamanho e conteúdo da sequência podem ser alterados através da chamada de certos métodos.

StringBuilder -> Uma sequência de caracteres mutáveis. Essa classe fornece uma API compatível com StringBuffer, mas com a não-garantia de sincronização (o que a diferencia da StringBuffer). Essa classe é designada para uso como uma substituta para StringBuffer onde ela será usada por uma única thread (como geralmente acontece).

Sempre que possível **StringBuilder** é recomendada, já que é mais rápida que a StringBuffer na maioria das implementações.

STRING, STRINGBUILDER E STRINGBUFFER - QUAL DEVO USAR ?

Se quisermos alterar a mensagem de uma String através da concatenação com o operador "+" ? O que vai acontecer ??

```
public class Exemplo {  
    public static void main(String[] args) {  
        String msg = "a";  
        msg = msg + "bc";  
        System.out.println(msg);  
    }  
}
```

A cada concatenação, foi a criado de uma nova String com a sequência de caracteres "a" e "bc", na qual msg faz referência (após deixar de referenciar "a"). **Mas "a" ainda está na memória do computador, só que em um outro local.**

STRING, STRINGBUILDER E STRINGBUFFER - QUAL DEVO USAR ?

Vamos agora medir o tempo para fazer 100.000 concatenações usando a classe String e usando a classe StringBuilder

```
public class ConcatenacaoString {  
  
    public static void main(String[] args) {  
        String msg = "";  
        double x, x0, deltaX;  
  
        x0 = System.nanoTime();  
        for (int i = 0; i < 100000; i++) {  
            msg = msg + i;  
        }  
        x = System.nanoTime();  
        deltaX = (x - x0)/Math.pow(10, 9);  
        System.out.printf("Segundos passados: %.0f\n",  
            deltaX);  
    }  
}
```

Segundos passados: 7

```
public class ConcatenacaoStringBuilder {  
  
    public static void main(String[] args) {  
        StringBuffer msg = new StringBuffer();  
        double x, x0, deltaX;  
  
        x0 = System.nanoTime();  
        for (int i = 0; i < 100000; i++) {  
            msg.append(i);  
        }  
        x = System.nanoTime();  
        deltaX = (x - x0)/Math.pow(10, 9);  
        System.out.printf("Segundos passados:  
%.5f\n", deltaX);  
    }  
}
```

Segundos passados: 0,00642

STRING, STRINGBUILDER E STRINGBUFFER - QUAL DEVO USAR ?

```
class PerformanceExample
{
    static void concatenationWithString()
    {
        String objectOfString = "Ravi";
        for (int i = 0; i < 10000; i++)
        {
            objectOfString = objectOfString + "kant";
        }
    }

    static void concatenationWithStringBuffer()
    {
        StringBuilder objectOfStringBuilder = new StringBuilder("Ravi");
        for (int i = 0; i < 10000; i++)
        {
            objectOfStringBuilder.append("kant");
        }
    }

    public static void main(String[] args)
    {
        long startTime = System.currentTimeMillis();
        concatenationWithString();
        System.out.println("Time taken by String object = "+
            (System.currentTimeMillis()-startTime)+"ms");

        startTime = System.currentTimeMillis();
        concatenationWithStringBuffer();
        System.out.println("Time taken by StringBuilder = "+
            (System.currentTimeMillis()-startTime)+"ms");
    }
}
```

Output:

Time taken by String object = 142ms

Time taken by StringBuilder = 2ms

STRING, STRINGBUILDER E STRINGBUFFER - QUAL DEVO USAR ?

Conclusão:

A **String** deve ser usada sempre que sua aplicação **exigir poucas concatenações**, já que **Strings são imutáveis** e portanto sempre criarão um novo objeto na memória com o novo conteúdo acrescentado.

StringBuffer e **StringBuilder** devem ser utilizadas sempre que houver um número elevado de concatenações em sua aplicação, e se precisar converter para String para exibir uma mensagem ou armazenar em algum lugar, basta usar o método `toString()`.

A diferença entre StringBuffer e StringBuilder é a questão da sincronização, portanto a primeira é mais recomendada para se trabalhar com threads.

