Parte 3 – Strings

Fernando Barros, Karima Castro, Luís Cordeiro,
Marília Curado, Nuno Pimenta

Os conteúdos desta apresentação baseiam-se nos materiais produzidos por António José Mendes para a unidade curricular de Programação Orientada a Objetos.

Quaisquer erros introduzidos são da inteira responsabilidade dos autores.

Strings

- As Strings, ou cadeias de caracteres, são tratadas em Java pela classe **String,** que pertence a **java.lang**
  - O conceito de classe será definido mais à frente

- Para criar uma String podemos fazer:

```
String nome = new String("Boa tarde");
```

- Dado que a utilização de Strings é muito comum, o Java permite uma abreviatura:

```
String nome = "Boa tarde";
```

- As Strings em Java são **imutáveis**, ou seja uma vez que lhe seja atribuído um valor, este não pode ser alterado

Strings

- É possível declarar uma variável do tipo String sem inicializá-la com um valor
- null ≠ String vazia
  - Uma string vazia é uma instancia de String de tamanho zero: ""
  - Uma string null não tem valor nenhum

Strings

- Exemplo:

```
String str1 = null;  //String NULL
String str2 = "";    //String vazia
String str3 = "  ";  //Nem NULL nem vazia
```

Strings

- Um caracter numa String é referido pela sua posição, ou index (semelhante ao tratamento de elementos numa tabela)
- O index do primeiro caracter é zero
- A **classe String** inclui um vasto **conjunto de métodos** que permitem diversas manipulações de objetos deste tipo
- Exemplos:
  - **length** - devolve o número de caracteres da String
  - **substring** - devolve uma sub String da original
  - **toLowerCase** - devolve uma String igual à original com todos os caracteres convertidos para minúsculas
  - **charAt** - devolve o caracter da posição fornecida da String original
  - **equals** - compara duas Strings, a original com a fornecida
  - **toCharArray** - devolve um array de caracteres correspondentes à String

Strings

- Exemplo:

```
String frase = "Hoje é quinta feira";
int tamanho = frase.length();                  // tamanho = 19
String palavra = frase.substring(0, 4);        // palavra = "Hoje"
palavra = palavra.toLowerCase();               // palavra = "hoje"
char c = palavra.charAt(2);                    // c = 'j'
boolean b = palavra.equals("Hoje");            // b = false
int indice = frase.indexOf("qui");             // indice = 7
palavra = frase.substring(indice, índice+7); // palavra = "quinta "
palavra = palavra.trim();                      // palavra = "quinta"
// tabela com os chars da String frase
char[] letras = frase.toCharArray();
letras[7] = 'Q';                               // 'q' → 'Q'
String frase2 = new String(letras);            // nova String'
```

Strings

- É importante referir que, caso produzam alguma alteração, os métodos que à primeira vista alteram a String (como **toLowerCase**) na realidade devolvem uma nova String deixando a original intacta
- Isto decorre de as Strings serem objetos imutáveis, pelo que não é possível a sua alteração
- Em vez disso os métodos devolvem um novo objeto da mesma classe

Strings

- Exemplo: Programa que escreve as iniciais de três palavras

```
class DEI {
  public static void main(String[] args) {
     String first = "Departamento";
     String middle = "Engenharia";
     String last = "Informática";

     String firstInit = first.substring(0,1);
     String middleInit = middle.substring(0,1);
     String lastInit = last.substring(0,1);
     String initials = firstInit.concat(middleInit);
     initials = initials.concat(lastInit);
     System.out.println(initials);
  }
}
```

Strings

- As instruções:

```
initials = firstInit.concat(middleInit);
initials = initials.concat(lastInit);
```

- poderiam ser substituídas por:

```
initials = firstInit.concat(middleInit).concat(lastInit);
```

Strings

- Funcionamento:
  - A mensagem **concat** é enviada a firstInit (com middleInit como argumento)
  - O objeto firstInit devolve uma referência para um novo objeto da classe **String** ("DE")
  - A mensagem **concat** é enviada ao novo objeto (com lastInit como argumento)
  - A referência para um novo objeto ("DEI") é devolvida e armazenada em initials
- A este tipo de estrutura dá-se o nome de **cascata**

Strings

- Em alternativa poderíamos usar:

```
initials = firstInit.concat(middleInit.concat(lastInit));
```

- Funcionamento:
  - A mensagem **concat** é enviada a middleInit (com lastInit como argumento)
  - O objeto middleInit devolve uma referência para um novo objeto da classe **String** ("EI")
  - A mensagem **concat** é enviada ao objeto firstInit (com a referência anterior como argumento)
  - A referência para um novo objeto ("DEI") é devolvida e armazenada em initials
- A este tipo de estrutura dá-se o nome de **composição**

Strings

- Numa **cascata** os resultados das mensagens são usados como recetores de mensagens adicionais, enquanto que numa **composição** os resultados das mensagens são usados como argumentos de outras mensagens
- Levado a um extremo a utilização de cascatas e/ou composições pode levar a código difícil de ler e propenso a erros
- Se utilizadas com cuidado e de forma contida, estas técnicas podem levar à escrita de programas mais claros e simples

Strings

- Como já vimos, a classe **String** tem um método **substring** que recebe dois inteiros. Por exemplo:

```
String frase = "Hoje é quinta feira";
String palavra = frase.substring(0, 4); //palavra="Hoje"
```

- No entanto, na mesma classe existe um outro método substring que recebe apenas um inteiro. Por exemplo:

```
String frase = "Hoje é quinta feira";
String palavra = frase.substring(14); //palavra="feira"
```

- Os dois métodos são diferentes e distinguem-se pelos parâmetros que requerem, ou seja são distintos porque têm uma **assinatura** diferente (a assinatura de um método é constituída pelo seu nome e pelos parâmetros requeridos)

Strings

- Os dois métodos **substring** da classe **String** são distintos e modelam comportamentos distintos (ainda que relacionados)
- Métodos com o mesmo nome, mas assinaturas diferentes na mesma classe dizem-se "**overloaded**"
- A prática de desenhar classes utilizando esta técnica chama-se "**overloading**"
- *Overload* de métodos será discutido mais a frente

Strings

- Outro método útil da classe String é o **split**
- Recebe um parâmetro correspondente a uma String "delimitadora"
- Devolve um array de strings (referências para objetos String), cada uma das quais correspondente a uma das substrings da string original, delimitadas pelo início ou fim (da string original), ou String "delimitadora"

Strings

- Exemplo:

```
public static void splitDemo() {
   String strSplit = "Metodo split de string";
   String[] str = strSplit.split(" ");
   for (String s: str)
       System.out.println(s);
}
```