Programming challenges in Java

2. Strings and output format

Nacho Iborra

IES San Vicente

Table of Contents

Programming challenges in Java

1. Strings manipulation

If we want to work with strings in Java, we have the *String* class, with some useful methods that we can use (convert to upper or lower case, get a substring, find a text...).

- We can **concatenate** strings with the '+' operator, or in some cases with '+=' operator (if we want to add a string at the end of another).
- We can't treat a string as a char array (as in C++ or C#), and get to each character with the corresponding index. If we want to get the character at a given position, we need to use the ***charAt*** method. We can get the length of a string with its ***length*** method.

```
for (int i = 0; i < text.length(); i++)
    System.out.println(text.charAt(i));
```

- We can compare two strings in some different ways:
  - If we want to know which is greater or lower, we can use the ***compareTo*** method. It returns a negative number if the string on the left is lower, 0 if both strings are equal, or a positive number if the string on the right is lower.

```
if (text1.compareTo(text2) < 0)
   System.out.println("Second text is greater");
```

  - If we want to see if two strings are equals, we use the ***equals*** method (remember, que must NOT use the == comparator for this purpose).

```
if (text1.equals(text2))
   System.out.println("Texts are equal");
```

- We can convert the whole char to upper and lower case with ***toUpperCase*** and ***toLowerCase*** methods:

```
String text = "Hello world";
String textUpper = text.toUpperCase();
```

- We can find a text in a string in many different ways:
  - If we only want to know if a text contains a given subtext, we can use ***contains*** method, that returns a boolean:

```
if (text1.contains("hello"))
   System.out.println("There is a 'hello' in the text");
```

  - If we want to know the index at which a given subtext appears, we can choose among ***indexOf*** (gets the first occurrence of the subtext, or -1 if it does not exist) or ***lastIndexOf*** (gets the last occurrence of the subtext, or -1 if it does not exist)

```
if (text1.indexOf("hello") < 0)
   System.out.println("There is NOT a 'hello' in the text");
```

  - If we want to know if a text starts with a given prefix or ends with a given suffix, we use the ***startsWith*** or ***endsWith*** methods, which return a boolean

```
if (text1.startsWith("Hello"))
   System.out.println("Text starts with 'Hello'");
```

- We can get a substring of a given string with the ***substring*** method. It has two parameters: the index from which we must start getting the substring (starting by 0), and the index at which we must stop getting the string (excluded). If this second

parameter is omitted, it returns the resulting string from the initial index to the end of the string. For instance, `"Welcome".substring(3, 5)` returns "co" (indexes 3 and 4 of the string).

- We can replace a substring with another one with the ***replace*** method. It has two arguments: the old text and the new text, and it returns the resulting string

  ```
  String result = text.replace("Hello", "Good morning");
  ```

  There are some other options for this purpose, such as ***replaceAll*** method, which uses regular expressions to match the text to be replaced, or ***replaceFirst***, which only replaces the first occurrence of the old text with the new one.

- We can split a String using a delimiter with the ***split*** method. It returns an array with the resulting parts.

  ```
  String text = "Hello world";
  String[] parts = text.split(" ");
  ```

1.1. Try yourself: Nana al bebé de papá y mamá

To practise with strings, let's try this challenge in Java.

IMPORTANT: as you can see, there is no condition to finish the input, so you must read from it with your *Scanner* object while there is something to read:

```
Scanner sc = new Scanner(System.in);
while(sc.hasNextLine())
{
    String text = sc.nextLine();
    ...
}
```

2. Formatting output

Apart from traditional `System.out.println` instruction to print data, we can use some other options if we want this data to have a given output format. To do this, we can use `System.out.printf` instruction instead of the previous one. This instruction behaves in a similar way than the original `printf` C function. It has a variable number of parameters, and the first of all is the string to be printed out. Then, this string can have some special characters inside it, which determine the data types that must replace these characters. For instance, if we use this instruction:

```
System.out.printf("The number is %d", number);
```

then the symbol *%d* will be replaced by the variable `number`, and this variable must be an integer (this is what *%d* means).

There are some other symbols to represent different data types. Here are some of them:

- *%d* for integer types (long, int)
- *%f* for real types (float and double)
- *%n* to represent a new line (similar to \n, but platform independent). In this case, we don't need to add a parameter at the end of *printf*.

We can place as many symbols as we want inside the output string, and then we will need to add the corresponding number of parameters at the end of the `printf` instruction. For instance:

```
System.out.printf("The average of %d and %d is %f", number1, number2, average);
```

2.1. Some format examples

Besides the primary simbols %d and %f, we can add some other information between the '%' and the letter, that specify some format information.

2.1.1. Specifying integer digits

For instance, if we want to output an integer with a given number of digits, we can do it this way:

```
System.out.printf("The number is %05d", number);
```

where the `05` means that the integer is going to have, at least, 5 digits, and if there are not enough digits in the number, then it will be filled with zeros. The output of this instruction if `number` is 33 would be `The number is 00033`. If you don't put the 0, then the number will be filled with whitespaces. So this instruction:

```
System.out.printf("The number is %10d", number);
```

if `number` is 33, it would produce the following output: `The number is         33`.

2.1.2. Specifying decimal digits

In the same way that we format integer numbers, we can format real numbers. We can use the same pattern seen before to specify the total number of integer digits:

```
System.out.printf("The number is %3f", number);
```

But, besides, we can specify the total number of decimal digits by adding a point and the total number desired, this way:

```
System.out.printf("The number is %3.3f", number);
```

Then, if `number` is 3.14159, the output would be `The number is  3.142`.

2.2. Try yourself: reloj a través del espejo

In order to check how to format the ouput in Java, you have this challenge ready to be implemented.