by Nacho Iborra

Development Environments
Block 2
Unit 3: Array and string management

2.3.1. Array management

As in many other languages, we use arrays in Java to deal with sets of data with a predefined size. Besides, all the elements of the array are of the same type, so that we can access each element by its index or position in the array.

2.3.1.1. Array declaration

We declare and use arrays in Java in a very similar way that we do in languages such as C#. However, Java only deals with two types of arrays:

Unidimensional arrays

They are declared and used like we do in C#:

```
int[] data = new int[10];  // Array to store 10 integer values
data[0] = 3;
```

Square brackets can also be placed after the variable name in the declaration:

```
int data[] ...
```

Bidimensional arrays and N-dimensional arrays

In this case, Java only accepts multidimensional arrays, this is, arrays that can have a different length in each row. We use these arrays as we do in C# (we can also place the square brackets after the variable name):

```
int[][] data2 = new int[5][]; // Array with 5 rows
data2[0] = new int[3];        // Row 1 has 3 columns
data2[1] = new int[10];       // Row 2 has 10 columns
...
data2[0][0] = 14;
```

If every row of the array is going to have the same number of columns, we can initialize the whole array at once:

```
int[][] data2 = new int[5][10];  // 5 rows, 10 columns
```

2.3.1.2. Declaring an array with initial values

If we want to declare an array and set its initial values, we can specify these values between curly braces in the declaration. The total number of values between curly braces will determine the length of the array

```
int[] someData = {10, 11, 12}; // Array size is 3
```

If we have a multidimensional array, we can specify the elements of each row in the same way:

```
int[][] someMoreData = {
    {1, 2, 3},
    {4, 5, 6, 7},
    {8, 9}
};
```

2.3.1.3. Exploring arrays

We can use the `length` property of the dimension we are currently exploring to determine its size:

```
for (int i = 0; i < data.length; i++)
        ...

for (int i = 0; i < data2.length; i++)
        for (int j = 0; j < data2[i].length; j++)
                ...
```

Proposed exercises:

2.3.1.1. Create a program called *MatrixAddition* that asks the user to enter two bidimensional matrices of 3 rows and columns, and then prints the result of adding them. Remember, in order to add two matrices, you must do it cell by cell:

```
result[i][j] = matrixA[i][j] + matrixB[i][j]
```

2.3.1.2. Create a program called *MarkCount* that asks the user to enter 10 marks (integers between 0 and 10). The program must output how many marks of each type have been typed. For instance, if the user types these marks: 1, 7, 5, 7, 2, 6, 7, 3, 5, 8, then the program must output:

```
Marks per category:
1: 1 marks
2: 1 marks
3: 1 marks
4: 0 marks
5: 2 marks
6: 1 marks
7: 3 marks
8: 1 marks
9: 0 marks
10: 0 marks
```

2.3.2. String 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...). You can have a whole list of available methods in the String official documentation. Let's explore some of them.

2.3.2.1. Creating and exploring strings

We can create a string in many different ways: with a constant value, asking the user to type something...

```
String text = "Hello world";
String name = scanner.nextLine();
```

We can also concatenate strings with the `+` operator, or in some cases with `+=` operator (if we want to add a string at the end of another).

```
text = text + ", how are you?";
```

We can't treat a string as a char array (as in C++ or C#), and get to each character with the corresponding index between square brackets. If we want to get the character at a given position, we need to use the `charAt` method. We can also get the length of a string with its `length` method.

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

2.3.2.2. Comparing strings

We can compare two strings (alphabetically) 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 check if two strings are equal, we use the `equals` method (remember, we must NOT use the `==` comparator for this purpose). We can also use `equalsIgnoreCase` method if we want to ignore if strings are in uppercase or lowercase.

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

if ("hello".equalsIgnoreCase("HELLO"))
    System.out.println("Text are equal ignoring cases");
```

2.3.2.3. Finding texts in strings

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 (text.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)

```
int pos = text.indexOf("hello");
if (pos >= 0)
        System.out.println("There is a 'hello' at position" + pos);
```

- 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 (text.startsWith("Hello"))
        System.out.println("Text starts with 'Hello'");
```

2.3.2.4. String conversions

We can convert the whole string to upper and lower case with `toUpperCase` and `toLowerCase` methods:

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

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. It replaces EVERY occurrence of the old string with the new 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(" ");
// Two parts, "Hello" and "world"
```

Proposed exercises:

2.3.2.1. Create a program called *SortJoin* that asks the user to enter a list of names separated by whitespaces. Then, the program must split the string, sort the names alphabetically and output them separated by commas. For instance, if the user types this name list: `Susan Kailey William John`, then the program must output `John, Kailey, Susan, William`.

2.3.2.2. Create a program called *CheckMessages* that asks the user to type 10 strings. The program must store them in an array, and then replace the text "Eclipse" with "NetBeans" in every string that contains "Eclipse". The program must output the updated version of the strings stored in the array, once the replacement has been done.

2.3.2.3 Create a program called *LispChecker*. LISP is a programming languages where every instruction is enclosed in parentheses. This could be a set of instructions in LISP:

```
(let ((new (x-point a y))))
```

> You must implement a program that takes a string with LISP instructions (just one string) and then check if the parentheses are correct (this is, the number of opening parentheses and closing parentheses are the same).

2.3.3. Some more challenges

Now that you know a little bit more about arrays and strings, let's try these challenges from *Acepta el reto*.

> **Proposed exercises**:
>
> **2.2.3.3.** Try to solve these challenges from *Acepta el reto* web site:
>
> - Queen ant's sandwiches
> - Lullaby for mom and dad's baby
> - Clock through a mirror

2.3.3.1. Challenges with "infinite" input

In some challenges, there is no condition to finish getting data from the input (such as Challenge 417). In these cases, you must read from the input with your `Scanner` object while there is something to read:

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