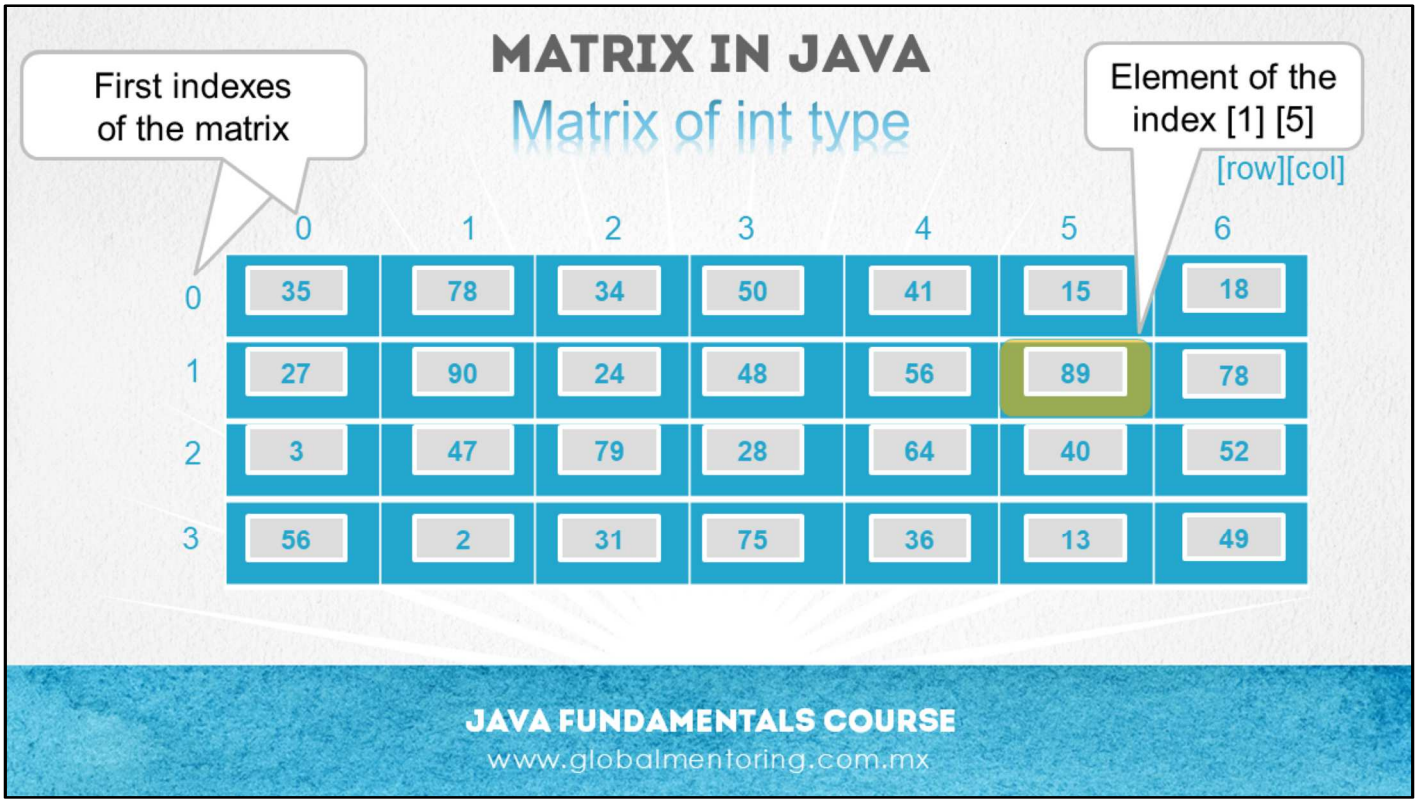
Hello, Ubaldo Acosta greets you. Welcome or welcome again. I hope you're ready to start with this lesson.

We are going to study the topic of Matrices in Java.

Are you ready? OK let's go!



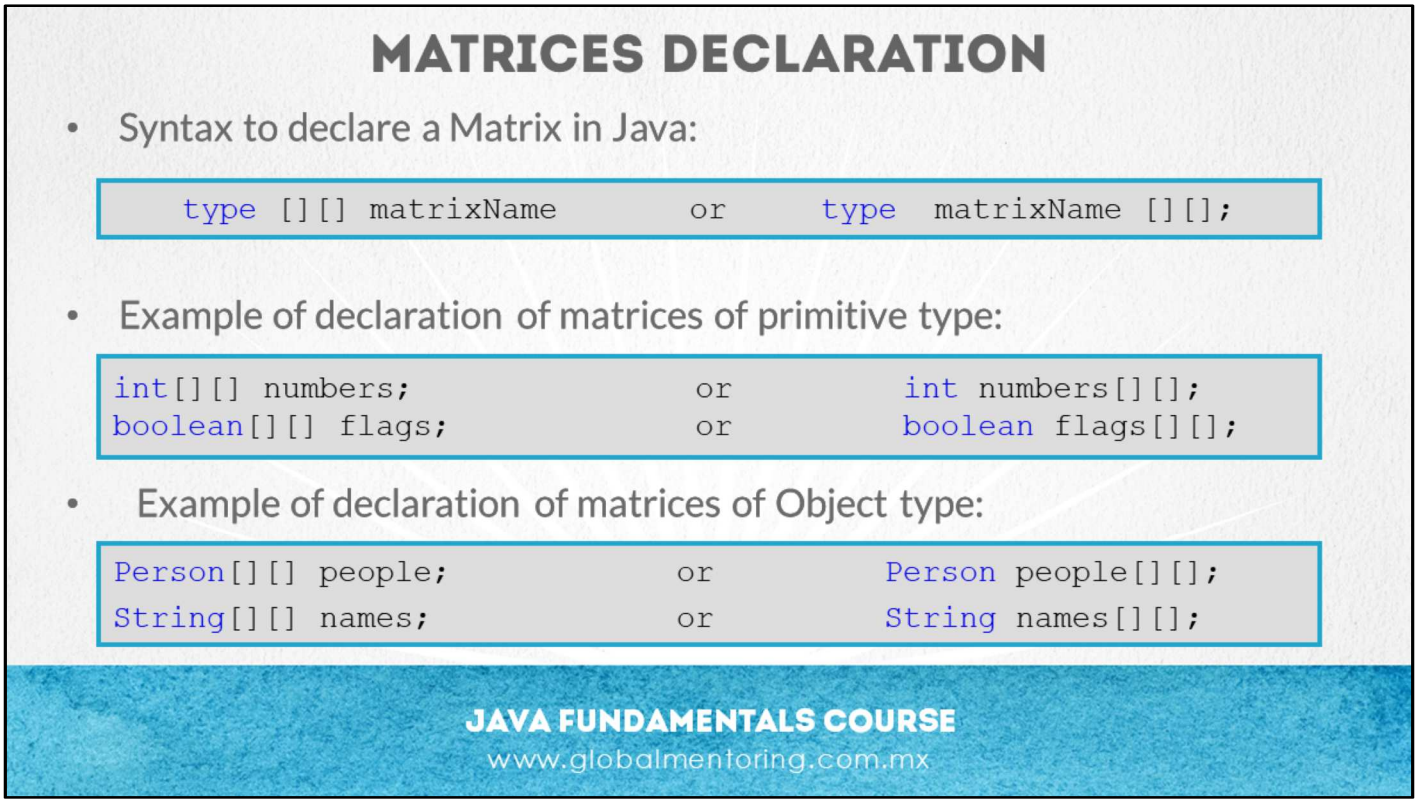
Another data structure in Java that we can use is a Matrix. Unlike a one-dimensional array, we could see that a matrix as two arrays, one array handles the rows, and another array the columns, and by putting them together we get a matrix. However, we will see that there are many similar things with a matrix, and once the concept of matrix is understood it is easier to understand the concept of matrix.

In the figure we can see a matrix of 4 rows by 7 columns, of integer type, however it can be of any type that we define.

We can retrieve the length of the rows with the code name Matrix.length and we can obtain the length of the columns by writing name Matrix[0].length, that is, with any valid row selected we can obtain the length of the columns. This will serve us later to iterate through in a nested way each of the elements of the matrix.

In the same way as an array, not all elements of a matrix must contain values. Values that do not have a value assigned will have the default value according to the data type defined for the matrix.

Next we will see the syntax to declare an array.

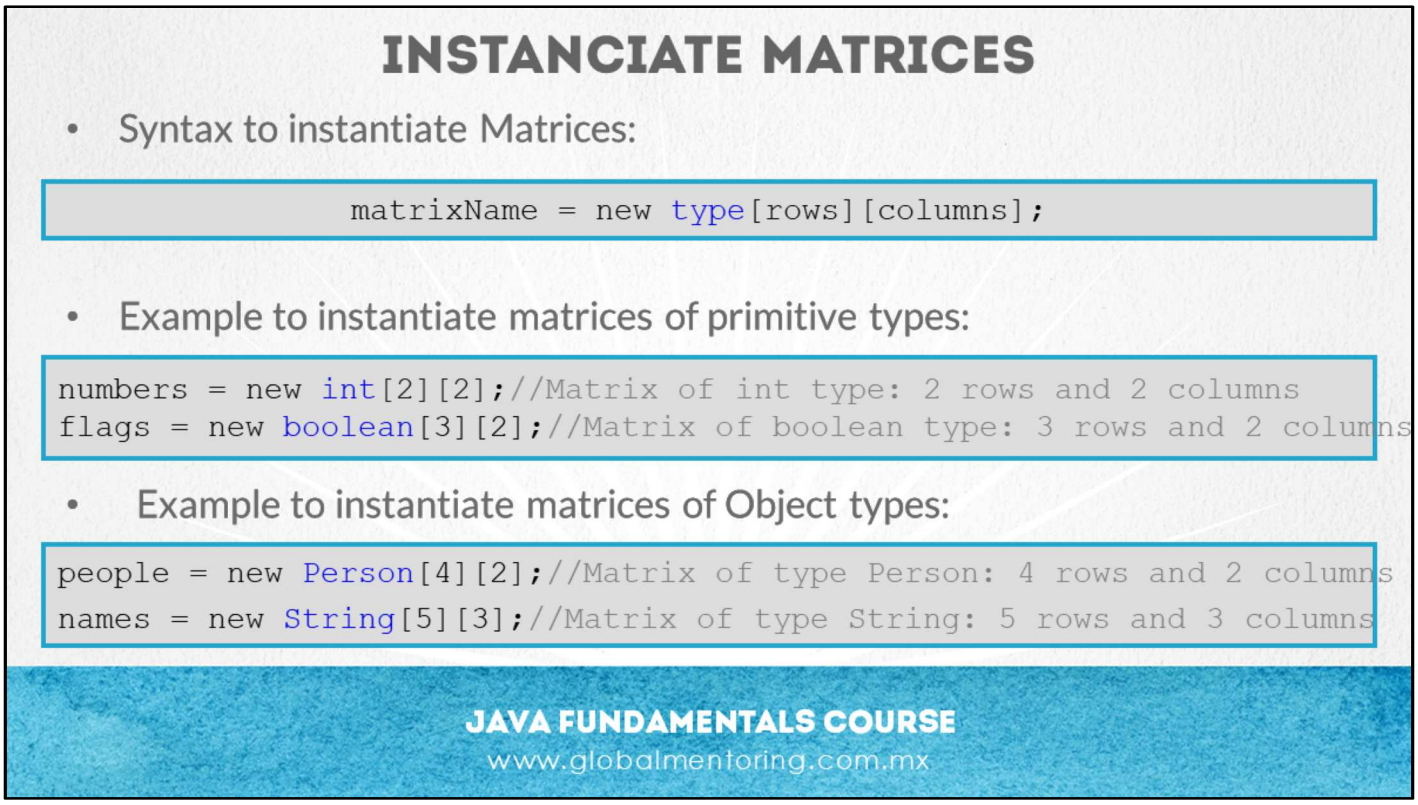


In this slide we can observe the syntax for the declaration of a matrix. Basically it is like declaring an array except that we are going to use double bracket [ ][ ], which can be used in two parts, either before the name of the variable or after the name of the variable, therefore the two options are shown in each statement.

Declare a matrix is equal to declare variables, we can declare matrices that store primitive types or store references to objects. In the slide we show both cases, first we show two examples of primitive type, one of type int and another of type boolean. Later we show the declaration of two matrices that will store references of objects of type Person and of type String.

Because a matrix is a collection of data, normally the name of a matrix is in plural, so that we can easily read the name of the variable, recognizing that it is a collection of data, although we will see later that it can be treated. not just arrays or matrices, but other data structures.

What is shown in the slide is only the declaration of the variable, we will see below how to initialize a matrix, since up to the moment with just declaring a variable of matrix type the JVM does not know how long and wide is the matrix, for it we must initialize it , let's see how.



Starting from the variables defined in the previous slide, in this slide we will see the syntax to instantiate a matrix according to the type of data we are using.

The syntax is very similar to instantiating a variable of type object, and in fact this is one of the features of Java, including arrays or any type in Java that stores a reference inherits from the Object class directly or indirectly, so both arrays and matrices also descend from the Object class.

However, the difference between instantiating a normal object and a matrix is that in a matrix we indicate the maximum number of elements that the matrix will contain, both in the number of rows and in the number of columns. Previously, in the definition of the variable, it was already indicated which type it is going to store, and now we must indicate that we will create an object of a certain type and that it will contain a number of rows and columns as indicated.

We can see in the slide several examples of initialization of matrices according to the type of data we choose. Next we will see how to initialize the elements of a matrix.

INITIALIZE THE ELEMENTS OF A MATRIX

- Syntax to initialize the elements of a Matrix:

```
nombreMatriz[índice_renglon][índice_columna] = valor;
```

- Example to initialize the elements of an integer array:

```
numbers[0][0] = 15; //The value of 15 is assigned to row=0 and column=0
numbers[1][0] = 13; //The value of 13 is assigned to row=1 and column=0
```

- Example to initialize the elements of an Object type matrix:

```
people[0][0] = new Person(); //The modified value is in row=0 and col=0
people[1][1] = new Person("Peter");//Value assigned in row=1 and col=1
nombres[0][0] = new String("Mary"); //Value assigned in row=0, col=0
nombres[2][1] = new String("Sara"); //Value assigned in row=2, col=1
```

In the slide we can observe the initialization of the elements of a matrix. What we must do to go adding elements to a matrix, is to select a row and a column with the respective indexes that we want to go initializing.

Therefore, it is important to know that unlike an array, in a matrix we will use two indexes to determine the position of an element, and that the first indexes of both the row and the column start at zero. It is also important to know that when we indicate a position first the row is indicated and then the column, always in that order. For example, the first element of a matrix will be the element [0] [0] and the length of an array is really two, the first will be determined by the nameMatrix.length which returns the row length, and later we can know the length of the columns by selecting any line, for example: nameName [0].length.

As in a matrix, we can only add elements up to the maximum of elements minus one, for example, if they are rows, it would be nameMatriz.lenght -1 and if it were the maximum of columns it would be nameMatriz [i] .lenght -1, where i is the line that is being worked on. If we exceed the maximum index in rows or columns and we want to add an element outside the maximum number of elements, it will throw an error, so we must know what the maximum number of elements is in both rows and columns.

We can see in the slide several examples of how to add elements to our matrix. We can add them manually, that is, one by one each element, or we can add the elements more dynamically using two element counters that have been added, both for the lines and for the columns, so that we can know if we have reached the limit of elements added or not.

In this slide we can see more clearly that not all the elements of a matrix will be always filled, for example if the integer matrix is 2 rows by 2 columns, then we have only filled 2 of the 4 available elements, this means that 2 elements will have the value by default, in this case the value of 0. Therefore, in many cases it will be convenient to have counters to know how many elements have been initialized in our matrix, which is different from the maximum number of elements that our matrix supports.

In the exercise of this lesson we will see how to initialize the elements of our matrices.

EXTRACT ELEMENTS FROM MATRICES

- Syntax to extract elements from matrices:

```
receivingVariable = matrixName[row_index][column_index];
```

- Example to extract the elements of a matrix of integer types:

```
int i = numbers[0][0]; //Extract the value in row 0 and column 0
int j = numbers[1][0]; //Extract the value in row 1 and column 0
```

- Example to extract the elements of a matrix of Object types:

```
Person p1 = people[0][0];//Extract the value in row 0 and column 0
Person p2 = people[1][0];//Extract the value in row 1 and column 0
String name1 = names[0][0];//Extract the value in row 0 and column 0
String name2 = names[1][1];//Extract the value in row 1 and column 1
```

To read or extract the elements stored in a matrix, simply indicate the name of the matrix and indicate two indexes, both the row and the column of the element that we want to extract, this will return the element of the indexes provided.

It is important not to exceed the maximum number of elements, both in rows and columns, otherwise an error will be returned.

We can observe in the slide several examples with matrices that store primitive types or Object types, in both cases the syntax is the same, we only have to have a variable that receives the value extracted from the respective matrix according to the specified indexes. Later we will see examples of how to extract the elements using a for loop to traverse each of the elements of a matrix.

DECLARATION, INSTANTIATION AND INITIALIZATION

- Syntax to declare, instantiate and initialize the elements of a matrix:

```
type [][] matrixName = {{value_list}, {value_list}};
```

- Example to declare, instantiate and initialize the elements of a matrix:

```
int[][] ages = {{10,23,41}, {10,23,41}, {10,23,41}, {10,23,41}};
```

- Example to declare, instantiate and initialize the elements of a matrix:

```
Persona[][] people = {{new Person(), new Person()}, {new Person(), new Person()}};
String[][] names = {{"Mary","Arthur","Sara"},{"Peter","Linda","Samantha"}};
```

JAVA FUNDAMENTALS COURSE
www.globalmentoring.com.mx

There is another way to declare matrices, and at the same time instantiate and initialize each of its elements. This is a different syntax in the way in which the values are assigned.

We can see in the slide several examples, both with primitive types, as well as Object types. And what we can observe is that in the same line of code the variable is declared, the matrix is instantiated and each of its values is initialized.

However, this syntax is not always possible to use since we would need to know in advance all the elements that are going to be stored in the matrix, and in many cases we do not have this information from the beginning, but if we do have this information before creating our matrix , then it is possible to use this simplified syntax.

EJEMPLO DE MANEJO DE MATRICES

Ejemplo de uso de matrices:

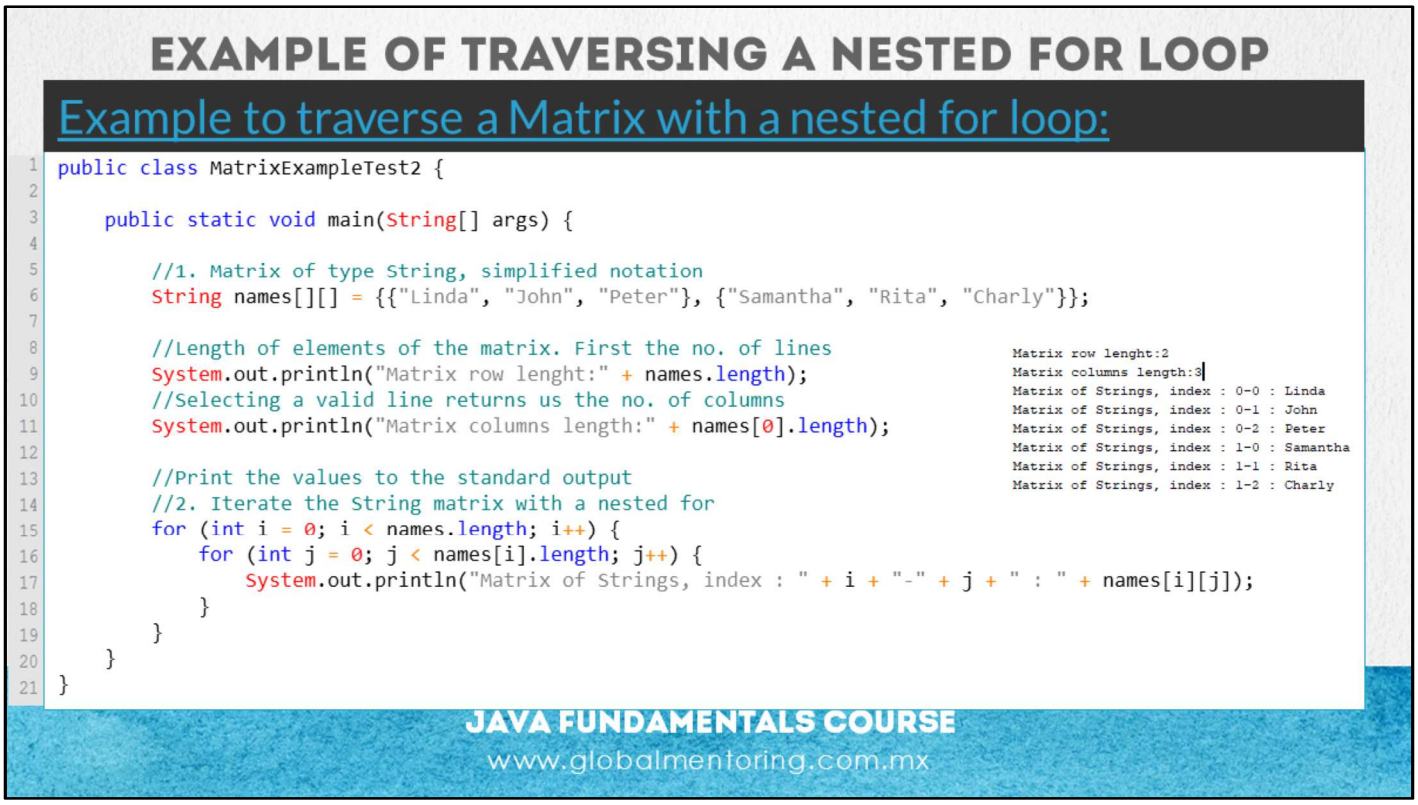
```
1  public class MatrixExampleTest {
2
3      public static void main(String[] args) {
4          //1. Declare a matrix of integers
5          int ages[][];
6          //2. instantiate a matrix of integers
7          ages = new int[3][2];
8          //3. Initialize the values of the matrix of integers
9          ages[0][0] = 30;
10         ages[0][1] = 15;
11         ages[1][0] = 20;
12         ages[1][1] = 45;
13         ages[2][0] = 5;
14         ages[2][1] = 38;
15
16         //Print the values to the standard output
17         //4. Read the values of each element of the matrix
18         System.out.println("Matrix of integers, index 0-0: " + ages[0][0]);
19         System.out.println("Matrix of integers, index 1-0: " + ages[1][0]);
20         System.out.println("Matrix of integers, index 1-1: " + ages[1][1]);
21         System.out.println("Matrix of integers, index 2-0: " + ages[2][0]);
22         System.out.println("Matrix of integers, index 2-1: " + ages[2][1]);
23     }
24 }
```


We can see in the slide an example for the use of matrices

From the declaration (line 5), the instantiation (line 7), the initialization of values (lines 9-14), and finally the reading of the values (lines 18-22).

We can see that in this case, all the values of the matrix are initialized, however if there were values that have not been initialized their value will be the default value of the declared type, in the case of the int type array the default value is 0 and in the case of the Persona type object, its default value will be null since it is an Object type.

Later we will carry out this exercise to put these concepts into practice.



Finally we will see an example with a matrix of type String using the simplified notation, and we will use it to show how to iterate the elements of an array with the help of a nested for loop. A nested for loop is simply a for loop within another for loop. In this case we will perform this nesting to be able to go through each of the elements of the matrix.

First of all we see an example of the use of simplified notation (line 6). In this case it is a matrix of type String, and in the same line we instantiate the matrix and initialize the values of it. In this case it is not necessary to indicate the number of rows or columns that the matrix will contain, this number will be obtained directly from the number of elements that are added in the initialization of the matrix. It should be noted that in this data structure it is not possible to make the matrix larger or smaller once declared or as in this case once initialized. However we will see in the next course the theme of collections, where we will see structures such as an ArrayList which are structures that can grow dynamically.

Once we have defined how many elements the matrix will have, we can obtain the no. of rows and columns of our matrix. So to check these values, we send to print the number of lines with the code names.length (line 9), and the number of columns with the code names [0] .length (line 11). It is therefore possible to combine two for loops, and using two counters, in this case the variables i and j, we will iterate each of the elements of the matrix.

The first loop, the outer one, runs through the rows of the matrix, and the innermost loop runs through the columns of the matrix. Therefore, in the output of our console, we will observe that the first 3 values the value of the rung index remains fixed, while the index of the column is moving until the columns for that selected rung have just been iterated. The end of the innermost loop will be when we have iterated all the columns for the selected row, according to the condition of the internal for loop: j <names [i] .length (line 16). Recall that the variable i controls the rows, and the variable j controls the columns. Finally, the most external loop will stop when all the rows of the matrix have been reviewed, according to the condition of the external for loop: i <names.length (line 15). With this we will have iterated all the rows, as well as each column of each selected row, and consequently all the elements of the matrix.

We will see later the execution of this code to put these concepts into practice.