A research laboratory grows a colony of bacteria within an area that can be considered as a grid surface measuring 30 x 30. Each box can be empty or contain a bacterium. From its initial configuration, the colony evolves generation after generation according to genetic laws that are described below and that depend on the number of neighbors that each box has:

- **Birth:** every empty box with exactly three neighbors will have a birth the next generation.
- **Death by loneliness:** every bacterium that occupies a box with 0 or 1 neighbors will die of loneliness the next generation.
- **Survival:** every bacterium that occupies a box with 2 or 3 neighbors will survive the next generation.
- **Death by suffocation:** every bacterium that occupies a box with more than 3 neighbors will die by suffocation the next generation.

Note that each bacterium has a maximum of 8 neighbors and that in the case of bacteria residing at the edges of the grid the number of neighbors is smaller.

The transition between generations is considered to be simultaneous in all squares of the colony. It is requested to design a program that simulates the evolution of the bacterial colony and determines, from a random initial situation, how many iterations are needed for the colony to reach a stable situation.

The program will look like this:

Program bacteria

Initialize variables

Generate initial generation

Show initial generation**While**

not stable situation **do**

Generate next generation

stable situation = (current generation = = next generation)

current generation = next generation

fMeanwhile

Show final generation

Show number of iterations performed

fProgram

Clarifications on some functions

The procedure **Generate initial generation** must create a colony at random. The easiest way is to fill the array with random values. Random r = new Random (); r.nextInt (2) and will return values that will be either 0 or 1 (you can consider for example that 1 represents a bacterium and 0 that it does not).

The procedure **Show initial generation** it will simply display the current status. There is no need to make jokes in the presentation, just to understand what is there. The procedure**Show final generation** it's exactly the same, it will just change the parameter we pass to it.

The procedure **Generate next generation** it creates the new generation following the rules explained before. Obviously to be able to do this you will need to call a function**neighbours** that indicates to us for each position of the matrix, how many neighbors there are.

We consider that the situation is stable when after creating a new generation there are no more changes in the colony.

This genetic system works, but due to possible programming errors it could be that it enters an infinite loop, so until you are sure that the program works and ends well you can also set as a condition of the loop that does not give more than 500 turns. It is also true that in some cases you can enter a two-lap cycle and therefore it is better to always put this condition.

If you are not sure where the program fails, you can view the dashboard inside the loop each time, but this will slow down the execution a lot. If you want to try it this way, do it with small matrices (6 x 6 or so). Once you are clear that it works, resize it.

NOTES: The program must be perfectly structured and the philosophy is as explained before. It doesn't have to be exactly that, but I don't think it can change the structure much.

Everything has to be done through functions (maybe **Initialize variables** no need) and you should think that the main program will not be much longer than the program on the previous page.

The code needs to be commented on.