Programming 1 – Exam 1
January 21, 2021

Submit the files `Prva.java`, `Druga.java`, `Tretja.java`, and `Cetrta.java`. You can test them as follows:

(1) `tj.exe Prva.java . .` (2) `tj.exe Druga.java . .` (3) `tj.exe` (4) `tj.exe`

(1) Write a program that reads an integer $n \in [1, 1000]$ and n integers from the interval $[1, 10^9]$ and, for each of these n integers, prints `DA` if its digits form a (not necessarily strictly) decreasing sequence and `NE` if this is not the case.

Every input integer is placed on its own line. The same should hold for the answers produced by your program.

Your program may only use the types `int` and `boolean`. In the opposite case, we will halve the number of points attained for this task!

Test case (input/output):

```
6
333
7210
2315432
1000000000
865312
9
```

```
DA
DA
NE
DA
NE
DA
```

(2) A long time ago, a gardener planted some trees inside a h-by-w rectangle (h is height, and w is width). By now, the rectangle has replicated itself infinitely many times in both dimensions.

A walker stands in the upper-left cell of the initial rectangle (this cell is always empty). Every hour, he moves a units downward and b units to the right and stops for a short time. How many hours is he going to walk in order to stop at k trees? He ends his journey after at most 10^4 hours, even if he visits less than k trees.

The first line contains integers $h \in [1, 100]$, $w \in [1, 100]$, $a \in [0, 10^4]$, $b \in [0, 10^4]$, and $k \in [1, 10^4]$. This line is followed by h lines containing w numbers 1 (tree) and 0 (empty space), which specify the initial rectangle. The numbers in the same line are separated by a single space. Your program should output the number of hours that the walker is going to walk.

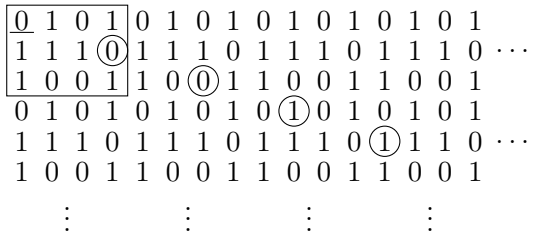
In 50% of the hidden test cases, the walker will not leave the initial rectangle.

Test case (input/output):

```
3 4 1 3 2
0 1 0 1
1 1 1 0
1 0 0 1
```

```
4
```

For an easier understanding of the example, take a look at the following picture (the visited cells are circled):



(3) You are given the following classes:

```
class Opravilo {    // a task, e.g., cutting wood, attaching legs, ...
    private String naziv;     // name
    private int zahtevnost;   // difficulty
}
class Projekt {     // a project, e.g., making a table
    private String naziv;      // name
    private Opravilo[] opravila;    // constituent tasks; length ⩾ 1
}
class Delavec {       // a worker
    private String ip;           // name and surname
    private int usposobljenost; // competence
}
class Delavnica {     // a workshop
    private Delavec[] delavci;     // the workers working in the workshop
}
```

The difficulty of a project is equal to the difficulty of its most difficult task. A worker can carry out a project if and only if its competence is equal to or greater than the difficulty of the project.

Write the following methods:

- [34%] `public int zahtevnost()` in the class `Projekt`:

  Returns the difficulty of `this` project.

- [32%] `public int univerzalci(Projekt[] projekti)` in the class `Delavnica`:

  Returns the number of workers in `this` workshop that could carry out, each on his/her own, all projects in the given array.

- [34%] `public boolean jePermutacijaOd(Object drugi)` in the class `Projekt`:

  Returns `true` if and only if the object `drugi` represents a project that contains *the same* tasks (the same objects!) as `this` project, but not necessarily in the same order. You may assume that there are no multiple pointers to the same task in the same project.

(4) The classes `Pravokotnik` (Rectangle) and `Krog` (Circle) are derived from the abstract class `Lik` (Shape). The class `Kvadrat` (Square) is derived from the class `Pravokotnik`. Every shape has its color (`Barva`). Colors are specified by components R (red), G (green), and B (blue). The natural order of shapes is determined by area: a shape A comes before a shape B if and only if A has a smaller area than B. To avoid working with the type `double`, round the area of the circle using the expression `(int) Math.round(...)`.

- [50%] Write the necessary code so that the method

  `public static void urediNaravno(List<Lik> liki)`

  naturally orders the given list of shapes.

- [50%] Write the necessary code so that the method

  ```
  public static Collection<Lik> poTipuInBarvi(Collection<Lik> liki)
  ```

  returns a collection in which the shapes from the collection `liki` are ordered by type (first rectangles that are not squares, then squares, and finally circles), and shapes of the same type are ordered by color. Colors have to be ordered first by increasing components R, then (for colors with equal components R) by increasing components B, and finally (for colors with equal components R and B) by increasing components B. **Beware:** the method must not change the collection `liki`!

In every hidden test case, each pair of shapes A and B differ in the sorting criterion. (For example, if shapes have to be naturally ordered, then no two shapes will have the same area.)