Home task № 5.3

Course

Programming Using Java

TOPIC: MULTIDIMENSIONAL ARRAYS

Task 1

Build an array, the elements of which are squares of the corresponding indexes. Output the result to the screen.

Task 2

Create an array of 20 random numbers in the range from -10 to 30. Write a program that determines the sum of array elements in the array after the first negative element (the first negative element must not be included in the sum). Output the obtained array and the sum to the console.

Task 3

Create an array of 200 random numbers in the range from 0 to 200. Determine the amount of one-figure, two-figure, and three-figure numbers percentage-wise. Output the obtained array and the number by units.

Example:

```
digit 1 = 4
digit 2 = 45
digit 3 = 39
```

Task 4

Create an array of 100 random numbers in the range from -300 to 555. Write a program that copies one array to another in the following way: first, all elements greater than 0, then all elements equal to 0, then all elements less than 0 are copied successively. Output the source array. Output the resulting array.

Task 5

Create an array of 20 random numbers in the range from -10 to 20. Determine the maximum number of consecutive positive elements that are not interrupted by zeros or negative numbers. Output the source array and the found fragment to the console.

Task 6

Create a square array of the *n* dimension filled with random numbers, output the array to the screen in a table, find the smallest and the biggest array element and output them to the screen (if there are several equal elements, output the row and column indexes of repetitions). Output search time in milliseconds. Array dimensions must be set from the keyboard.

Task 7

Fill the *n* dimension square array with ascending numbers, in the S shape. Output the result to the screen in compliance with the columns width.

Example:

```
 1  2  3  4
 8  7  6  5
 9 10 11 12
16 15 14 13
```

Task 8

Fill the *n* dimension square array with numbers that increase by 1 spiral-wise (the *n* number must be set from the keyboard). Output the result to the screen in compliance with the columns width. For example, the 4x4 array must look as in the example.

Example:

```
 1  2  3  4
12 13 14  5
11 16 15  6
10  9  8  7
```

Additionally: do this task using only one array + 1 point.

Task 9

There are numbers **n** and **m**. Create an array of the dimensionality **[n][m]** and fill it with numbers diagonally as shown in the example. Output the result to the screen in compliance with the columns width.

1	2	4	7
3	5	8	11
6	9	12	15
10	13	16	18
14	17	19	20

Task 10

> There are n rows of m seats in a cinema (m and n must be set from the keyboard). Information on the sold tickets is stored in a two-figure array of the m by n dimensionality. Number 1 means that the ticket for the given seat is already sold, number 0 means that the seat is vacant. There is a request on selling k tickets for neighboring seats in one row (k must be set from the keyboard).

Determine whether it is possible to comply with this request. If there are solutions, output numbers of rows and numbers of vacant seats for selling.

The number of taken seats and what seats are taken at the moment of request should be determined randomly.

Output the source array to the screen.

Task 11

Enter from the keyboard an ordinal number in the range from 0 to 1 000 000 inclusive. Write them in letters. Mind that ordinal numbers have different endings, for example, twenty-third, one hundred and sixty-fifth, etc.

For example, when entering 1125, the program must output to the console “one thousand and twenty-fifth”.

Task 12

Fill a square array of the *n* size with the knight's move (the knight's move is L shaped). There is a simple algorithm that allows filling the board with the knight's move of the size from 5 to 70. That is, fill the array with number 0 first, and then put number 1 on the first square (array element), 2 on the next one, where the knight steps, and so on until there is no squares on which the knight haven't stepped. If there are zero values, the algorithm works incorrectly. Output the array to the screen. The array dimensionality must be set from the keyboard (*See*. The knight's tour).

> ***It is recommended to use the Warnsdorf's rule to solve this problem.***