Stream Programming - 2019
Laboratory 2

Task 1 Create a class `PrimeNumber` with one constructor `PrimeNumber(val n:int)` and a function `calculatePrimeNumbers(val n:int)` which calculates all prime numbers within the range between 2 and n. Then, implement the public function `number(val m:int):int` which returns the mth prime number. The function should work properly for the numbers from 0 to k, where k is the amount of prime numbers within the range between 2 and n.

Create object `Test` with the method `main` which for the first argument creates an array of the prime numbers and then for the next arguments prints the values of the calculated prime numbers from that array.

In the case of a wrong first argument, the program should print an error message and finish the work. In the case of incorrect remaining arguments, it should write, instead of the value, information about this error and go to the calculation for the next argument.

The example of program execution:

```
> scala Test 14 0 2 4 8 -1 aaaa
0 - 2
2 - 5
4 - 11
8 - out of range number
-1 - out of range number
aaaa - invalid argument

> scala Test -4 8
-4 - out of range number
```

Task 2 Create a class `PascalTriangleRow` with one constructor `PascalTriangleRow(val n:int)` and a function `calculate(val n:int)`. The function should calculate nth line of the Pascal triangle and write it into an array. Then implement the public function`factor (val m:int):int` returning the mth item of the Pascal's triangle line n. This function should work properly for numbers from 0 to n (constructor argument).

Create an object `Test` with a method `main` which for the first argument creates a Pascal's triangle line and then for the next arguments prints the appropriate values of this line.

The example of program execution:

```
> scala Test 4 0 1 3 8 -1 aaaa
0 - 1
1 - 4
2 - 6
8 - out of range number
-1 - out of range number
aaaa - invalid argument

> scala Test -4 8
-4 - invalid row number
```

Hint 1 In both tasks you should declare an array. Here are the examples of two equivalent array declarations. Each array can hold at most three values of the type String.

```
var z:Array[String] = new Array[String](3)
```

```
var z1 = new Array[String](3)
```

Now, you can write some values to the arrays:

```
z(0) = "Zara"; z(1) = "Nuha"; z(2) = "Ayan"
```

There is one more way of defining an array:

```
var z = Array("Zara", "Nuha", "Ayan")
```

The following example presents multi-dimensional array processing:

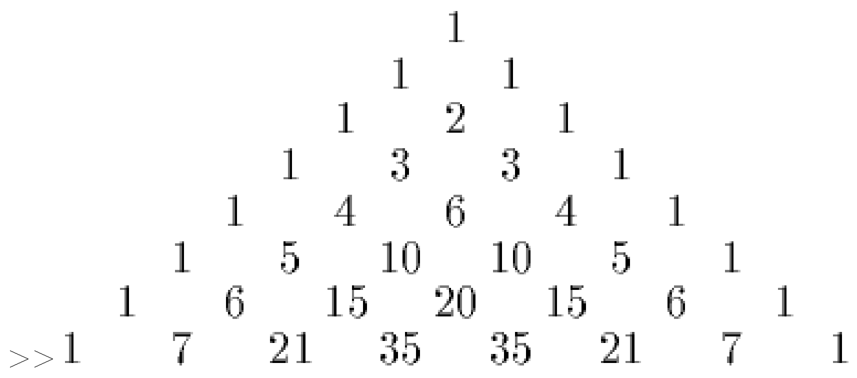
```
import Array._
object Demo {
   def main(args: Array[String]) {
      var myMatrix = ofDim[Int](3,3)
      // build a matrix
      for (i <- 0 to 2) {
         for ( j <- 0 to 2) {
            myMatrix(i)(j) = j;
         }
      }
      // Print two dimensional array
      for (i <- 0 to 2) {
         for ( j <- 0 to 2) {
            print(" " + myMatrix(i)(j));
         }
         println();
      }
   }
}
```

In the beginning, the multi-dimensional array is declared. Then, its elements are populated with the numbers from 0 to 2 (first `for` loop). At last, the elements are printed (second `for` loop). The output would be as follows:

```
0 1 2
0 1 2
0 1 2
```

Hint 2 Pascal's triangle. The rows of Pascal's triangle are conventionally enumerated starting with row n = 0 at the top (the 0th row). The entries in each row are numbered from the left beginning with k = 0 and are usually staggered relative to the numbers in the adjacent rows. The triangle may be constructed in the following manner: In row 0 (the topmost row), there is a unique nonzero entry 1. Each entry of each subsequent row is constructed by adding the number above and to the left with the number above and to the right, treating blank entries as 0. For example, the initial number in the first (or any other) row is 1 (the sum of 0 and 1), whereas the numbers 1 and 3 in the third row are added to produce the number 4 in the fourth row.

Figure 1 shows the Pascal's triangle of 7 rows.



```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5  10  10   5   1
        1   6  15  20  15   6   1
>>    1   7  21  35  35  21   7   1
```

Rysunek 1: Pascal's triangle