Problem 1

Write a static *recursive* function which takes an array of **int**s, starting index and a value; the function returns (but doesn't print!) number of occurrences of this value in the part of the array starting from the given index; for example, the following program

download *Recurs1.java*

```
public class Recurs1 {
   static int count(int[] arr, int from, int what) {
       // ...
   }

   public static void main(String[] args) {
       int[] a = {2,3,2,4,3,1,6,3,2,3};
       System.out.println("2 -> " + count(a,0,2));
       System.out.println("3 -> " + count(a,0,3));
   }
}
```

should print

```
2 -> 3
3 -> 4
```

Do not use static fields, loops or additional arrays/collections/strings!

Problem 2

Write the function

```
static void rotateMatrix(int[][] a)
```

which given a **square** array of **int**s rotates this array by 90° clockwise; for example, for the matrix

```
1 2 3 4
5 6 7 8
9 7 5 3
8 6 4 2
```

the result should be

```
8 9 5 1
6 7 6 2
4 5 7 3
2 3 8 4
```

Using auxiliary arrays, collections or **String**s is *not* allowed.

Additionally, define the function

```
static void printMatrix(int[][] a)
```

which pretty-prints two-dimensional arrays (one row in one line), so the program

download *RotMatrix.java*

```
public class RotMatrix {
    public static void main(String[] args) {
        int[][] a = { {1, 2, 3, 4},
                      {5, 6, 7, 8},
                      {9, 7, 5, 3},
                      {8, 6, 4, 2} };

        int[][] b = { {1, 2, 3, 4, 5},
                      {5, 6, 7, 8, 9},
                      {9, 7, 5, 3, 1},
                      {8, 6, 4, 2, 0},
                      {0, 4, 6, 4, 0} };
        System.out.println("Array a - original");
        printMatrix(a);
        rotateMatrix(a);
        System.out.println("Array a - rotated");
        printMatrix(a);

        System.out.println("Array b - original");
        printMatrix(b);
        rotateMatrix(b);
        System.out.println("Array b - rotated");
        printMatrix(b);

    }
    //
    // ... implementation of the functions
    //
}
```

prints

```
Array a - original
1 2 3 4
5 6 7 8
9 7 5 3
8 6 4 2
Array a - rotated
8 9 5 1
6 7 6 2
4 5 7 3
```

```
2 3 8 4
Array b - original
1 2 3 4 5
5 6 7 8 9
9 7 5 3 1
8 6 4 2 0
0 4 6 4 0
Array b - rotated
0 8 9 5 1
4 6 7 6 2
6 4 5 7 3
4 2 3 8 4
0 0 1 9 5
```

Problem 3

Create a class **FuncStat** containing only public static functions:

- `public static int fiboR(int n)` calculating the n-th Fibonacci number

$$F_n = \begin{cases} n & \text{for } 0 \leqslant n < 2, \\ F_{n-1} + F_{n-2} & \text{for } n \geqslant 2 \end{cases}$$

  using this recursive formula, which is rather unwise but enlightening — therefore, the function should be *recursive* (in particular, no loops are allowed in its implementation!);
- `public static int fiboI(int n)` calculating the n-th Fibonacci number without recursion (i.e., iteratively — using a loop);
- `public static int factR(int n)` calculating $n!$ recursively;
- `public static int factI(int n)` calculating $n!$ iteratively;
- `public static int gcdR(int a, int b)` calculating the GCD, i.e., the greatest common divisor, of a and b recursively;
- `public static int gcdI(int a, int b)` calculating the GCD of a and b iteratively;
- `public static int maxElem(int[] arr, int from)` returning the largest element of the elements of array arr starting from element with index from. It must be a **recursive** function. Function **max** from class **Math** may be useful but is not necessary;
- `public static int numEven(int[] arr, int from)` returning the number of even elements of array arr starting from element with index from. It must be a **recursive** function.
- `public static void reverse(int[] arr, int from)` reversing the order of elements of the array arr starting from the element with index from. It must be a **recursive** function. Do *not* create any auxiliary arrays!.
- `public static boolean isPalindrom(String s)` returning **true** if, and only if, string s is a palindrom, i.e., a word which reads the same forward and bac-

kward, as, e.g., words *radar* or *madam*. Methods **charAt** and **substring** from class **String** may be useful. It must be a **recursive** function.

Then, in the **main** function *of a separate class* **Main** test all these functions.

Remark: according to Euclid (*Elements*, Book VII), the greatest common divisor of two positive integers, a and b, can be calculated as follows:

1. if $a = b$, then the result is a (or b, as they are equal);
2. from the larger of these two numbers subtract the smaller and go to 1.

Do not use any classes from packages other than basic ***java.lang*** (in particular, no collections are allowed).

You can assume that functions will be invoked with legal arguments (e.g., no negative argument of the factorial function).

For example, the following **main** function

download *FuncStat.java*

```
import java.util.Arrays; // for printing
    // ...
    public static void main (String[] args) {
        System.out.println("Wait...");
        System.out.println(FuncStat.fiboR(45));
        System.out.println(FuncStat.fiboI(45));
        System.out.println(FuncStat.factR(12));
        System.out.println(FuncStat.factI(12));
        System.out.println(FuncStat.gcdR(12125,40643));
        System.out.println(FuncStat.gcdI(12125,40643));
        int[] a = {3,8,2,9,7,4};
        System.out.println("Max     : " + FuncStat.maxElem(a,0));
        System.out.println("Num even: " + FuncStat.numEven(a,0));
        System.out.println("Before:   " + Arrays.toString(a));
        FuncStat.reverse(a,0);
        System.out.println("After :   " + Arrays.toString(a));
        System.out.println("Is 'radar' a palindrom? " +
                           FuncStat.isPalindrom("radar"));
        System.out.println("Is 'abba' a palindrom?  " +
                           FuncStat.isPalindrom("abba"));
        System.out.println("Is 'rover' a palindrom? " +
                           FuncStat.isPalindrom("rover"));
    }
```

should print (note that calculating Fibonacci number F_{45} recursively takes a while...)

```
Wait...
1134903170
1134903170
479001600
479001600
```

```
97
97
Max     : 9
Num even: 3
Before:   [3, 8, 2, 9, 7, 4]
After :   [4, 7, 9, 2, 8, 3]
Is 'radar' a palindrom? true
Is 'abba' a palindrom?  true
Is 'rover' a palindrom? false
```

Problem 4

Write a static function **isLess** which takes two strings (**String**) and returns a value of type **boolean**: **true** if, and only if the first string is strictly "smaller" than the second. The criteria of comparison are:

- shorter string is "smaller" than a longer (the length of a string `s` is `s.length()`);
- if the strings are of the same length, the one lexicographically *later* is "smaller".

To compare string lexicographically, you can use the method **compareTo** from class **String**: if `s1` and `s2` are references to objects of class **String**, then

```
s1.compareTo(s2)
```

returns a *negative* integral value if `s1` is lexicographically earlier than `s2`, a *positive* number, if `s2` is earlier, and 0 if they are equal.

Write also a static function **sortSel** which sorts an array of strings passed as the argument and using the selection sort algorithm; use your function **isLess** to compare strings.

For example, a program with the following **main** function

download *StringCmp.java*

```
import java.util.Arrays;
public class StringCmp {
    // ...
    public static void main (String[] args) {
       String[] arr = {"Kate", "Bea", "Mary", "Bea", "Zoe"};
       System.out.println(Arrays.toString(arr));
       sortSel(arr);
       System.out.println(Arrays.toString(arr));

    }
}
```

should print

```
[Kate, Bea, Mary, Bea, Zoe]
[Zoe, Bea, Bea, Mary, Kate]
```