Keep your programs neat from the start on:

- Use clear, sensible names for variables etc.
- Let class names always begin with an uppercase letter, variable names with a lowercase letter; use all capitals for constants.
- Use indentation rules consistently. After an opening brace ({}), indent every line by two or more positions and put the closing brace (}) on a single line, not indented. See the program below for an example.
- Don’t make lines longer than the width of your screen; otherwise it is rather hard for the instructor to help you with your program. Don’t hesitate to break a line; in Java, a line can be broken anywhere a space could appear (except in strings).

1. Square root (linear and binary search)

The following program uses the so-called **linear search** technique to calculate the square root (wortel) of a positive integer number.

```java
import java.util.*; // provides Scanner class description

class SquareRoot {
    public class SquareRoot {
        // inputs a positive integer number, calculates and outputs
        // its square root rounded down
        public static void main(String[] args) {
            Scanner sc = new Scanner(System.in); // create Scanner object for console input
            int n; // to store input number
            int r; // to store approximation of square root of n

            // ask for and input a positive integer number
            System.out.println("Input a positive integer number: ");
            n = sc.nextInt();

            if (n > 0) { // calculate square root using linear search
                r = 0; // initialisation
                // invariant P: 0 <= r and r * r <= n
                while ((r + 1) * (r + 1) <= n) {
                    r = r + 1;
                    // P
                }
                // 0 <= r and r * r <= n < (r+1)* (r+1)
                System.out.println("The square root of "+n + " rounded down is "+ r + ".");
            } else {
                System.out.println("The input number "+n + " is not positive.");
            }
        }
    }
}
```

How many steps does the repetition (while-statement) take to calculate the result? Why does the repetition terminate?

The number of steps to calculate the approximation of the result can be reduced as follows. Initialize integer variables $r$ and $s$ to a lower bound (ondergrens) and an upper bound
(bovengrens) of the square root, respectively (\( r \times r \leq n < s \times s \)). As long as \( r \) and \( s \) are not next to each other (\( r + 1 \neq s \)), choose some integer number \( h \) between \( r \) and \( s \) (not equal to them). Test whether \( h \) is a lower bound for the square root, and if so assign \( h \) to \( r \), otherwise assign it to \( s \).

If we choose \( h \) to be \( r + 1 \), then we essentially end up with the linear search again. If we choose \( h \) to be \( s - 1 \), then we again get a linear search but this one is less efficient than the first one (it takes more steps in the repetition; why?). Choosing \( (r + s) / 2 \) for \( h \) we end up with the so-called binary search. In this case how many steps are needed to calculate the result? Why does the repetition terminate? Implement this technique by adapting the given program. Let the program also output the number of repetition steps needed for the given input. Run both the linear search and binary search program on a number of ever greater input numbers.

2. Counting letters and digits

(a) Develop a program that reads the text in a given input file tekst.txt and outputs for each of the letters 'a', ..., 'z', 'A', ..., 'Z' and for each of the digits '0', '1', ..., '9' the number of times it occurs in the text.

(b) Develop a program that reads a word (string) and outputs the number of times the word (string) occurs in text file tekst.txt.

3. Sorting

An array of integers is called sorted if its elements form an ascending sequence (i.e. array \( a \) is sorted if \( a[0] \leq a[1] \leq a[2] \leq \cdots \leq a[a.length - 1] \)). Rearranging the elements of an array such that it becomes sorted, is called sorting. One algorithm to sort an array is called selection sort and can be described as follows:

- repeatedly execute for the current subarray (which initially will be the whole array) as long as it contains more than one element
  - compute the position of a largest element in the current subarray
  - swap this largest element with the last element in the current subarray
  - reduce the subarray by omitting the last element
- after termination of this repetition the array is sorted

First, turn this informal description into Java code. Indicate what variables are needed and give an invariant for the outermost repetition (the code should contain a repetition nested within another repetition).

Develop a program that reads all numbers from a given input file numbers.txt and writes these numbers in ascending order (sorted) to output file numbers_sorted.txt. The program should use the selection sort algorithm described above. The file numbers.txt contains only numbers separated by whitespace.

4. Game of dice

In this exercise you are asked to develop a program that simulates a game of dice (to be described next) and prints winning statistics of the simulated game.

The game of dice is played by 6 players numbered 1 up to 6. The game consists of a number of rounds. In each round player 1 throws with one 12-sided die (dodecahedron) the sides of which are marked with different numbers of pips (ogen) in the range from 1 to 12; players 2 to 6 all throw with 2 regular (6-sided) dice. The round is won by the player with the highest score provided he is the only one with that score; otherwise there is no winner of the round. The winning statistics of a game consist of the number of rounds each of the players has won and the number of rounds that had no winner.
Analyze what variables and constants are needed in a program, and which tasks and subtasks there are. The program should be well structured and contain clear comments.

**hint** A random integer number in the range from 1 to $N$ (bounds included) can be generated by the following Java-expression:

```
(int) (Math.random() * N + 1)
```

where the call `Math.random()` generates a random double in the range 0.0 (inclusive) to 1.0 (exclusive).