1. Factorial

For a natural number \( N \) \((N \geq 0)\) its factorial is defined by \( N! = 1 \times 2 \times 3 \times \cdots \times N \) with the convention that \( 0! = 1 \). Develop a program (class) that inputs a nonnegative integer number, calculates its factorial, and outputs the number and its factorial (the program should be structured like the programs in the examples presented during the lectures). Implement the calculation of the factorial with a repetition using some additional variables. Carefully describe a property (properties) that hold(s) for these variables before and after each execution of the body of the repetition (phrased differently: what is the invariant?). Why does the repetition terminate after a finite number of steps (variant function)?

Calculation of the factorial is a subtask that can be delegated to a method. Give a variant of your program using a method with header `static void fac(int n) {...}` for the subtask (i.e. the method calculates the factorial of its integer parameter and assigns it to a global variable in the class for storing the factorial). Also, give a variant of your program using a method with result (function) with header `static long fac(int n) {...}` for the subtask (i.e. the method calculates the factorial of its integer parameter and returns it as its result).

2. Sorting (revisited)

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 have and use the following methods (NB use local variables and/or parameters whenever possible)
• a method `inputNumbers` that reads all numbers from a given file, and stores the integers in an array that is created by the method, but declared as a global (static) variable of the program (class)

• a method `printArray` that displays the current content of the array on the console output

• a method `selectionSort` that sorts the array using selection sort as described above

The `main`-method should implement the program by calling the above methods.

Add a method `transparentSelectionSort` that is a variant of the method `selectionSort` that displays the contents of the array after each step in the sorting process.

3. Stars

Write a class `Stars` with the following methods (subroutines) (the `main`-method of the class should consists of a number of calls to these methods)

(a) a method with header

```java
static void print10Stars()
```

that prints 10 stars on a separate line, like this

```
**********
```

(b) a method with header

```java
static void printStars(int n)
```

that prints `n` stars on a separate line.

Hint: the method call

```java
System.out.print("*");
```

prints one star without a newline. A newline can be printed by calling

```java
System.out.println();
```

(c) a method with header

```java
static void printRectangle(int h, int w)
```

that prints a rectangle of stars that is `h` stars high and `w` stars wide. The call

```java
printRectangle(3, 5);
```

should result in the output

```
2
```
(d) a method, similar to `printRectangle`, that prints a triangle of given size; you must decide for yourself how the triangle is specified, and how it is shaped and positioned.

(e) All of the above methods print only stars. Adapt the methods in such a way that the symbol to be used in printing can be specified at the moment of call.

4. Speaking of Money

You are going to develop a program that pronounces an amount of money in Dutch (or English, if you do not speak Dutch): given an amount in numbers, it should output the same amount in words. The amount is specified as a number of euros and a number of cents. For instance, when the input is 12 56, it will output "twaalf euro en zesenvijftig cent" or "twelve euros and fifty-six cents". The maximum amount to be handled is 1000000.99. The minimum amount is 0.00.

The program will repeatedly ask for an amount in the form of two integers, one representing the amount of euros and the other the amount of cents, until the word stop is given instead of the two integers. When an amount outside the specified interval is input, it will issue a message and ask again for an amount. Your program does not have to take other input into account.

Considerations

(a) First implement the following functions

```java
// @pre: 0 <= h < 20
// @return: Dutch text of number h (i.e., "een", "twee", etc.)
static String pronounce20(int h)

// @pre: 2 <= h < 10
// @return: Dutch text of number 10*h (i.e., "twintig", "dertig", etc.)
static String pronounce10s(int h)
```

You will need several additional functions `pronounce100`, `pronounce1000`, ... for pronouncing numbers up to 100, up to 1000, ..., as well as at least one method taking care of input, conversion, and output.

(b) Decompose the problem in subproblems and write a method for each of these subproblems. The functions typically take an integer, sometimes two, as parameter and produce a String with Dutch (or English) text. These functions should not modify (global) variables and should not deal with input or output. In other words, they are `side-effect free`. Formulate for every method what should hold before the call and what is assumed about the parameter(s) and what it returns.
(in case of a function) or what will hold after the call (in case of a procedure / pure method). Add this in comment before the method, preceded with @pre (for precondition), @returns (for return value), and @post (for postcondition). Use the functions mentioned above.

(c) To choose between the various parts of the text, such as "een", "twee", ..., it is not necessary to use the somewhat bulky construction of multiple if-statements or a switch-statement. Try to deal with this using one or more arrays of strings.