KNIME Coding Challenge (Java)

The following coding challenge is designed to check your Java development skills, especially for software design and efficient implementations.

You are free to choose libraries, data formats, names, etc. unless otherwise stated in the description below. However, you should be able to explain your decisions during the interview.

The challenge is to implement a small Java program with a command line interface. The program shall read a plaintext file, apply a pipeline of operations on the data and present some basic statistics. The data resulting from the last operation in the pipeline have to be written to a file or printed to standard out.

The Java program should support the following command line arguments:

```
java org.yourname.Main \
 --input <ascii-file-containing-strings-separated-by-linebreak.txt> \
 --inputtype <string|int|double> \
 --operations <operation1[,operation2[...]]> \
 --threads <n> \
 [--output <output.txt>]
```

where

- `--input` specifies a plaintext file (UTF8 encoded) containing the data elements, separated by newline, e.g.:

example.txt
foo
bar
42
Hello world

- `--inputtype` specifies the type of data elements. `string`, `int` and `double` need to be supported.
- `--operations` describes a pipeline of what to do with data elements. Each operation only applies to certain types of data. Implement at least the following operations:
  - `capitalize`: Capitalizes a string element.
  - `reverse`: Reverses the characters in a string, or the decimal string representation of an integer (1020 becomes 201).
  - `neg`: Negates an integer or double number (10 becomes -10).
- `--threads` specifies the number of threads to use when reading the input file and applying operations.
- `--output` optionally specifies the file to write the output to. If no output file is specified, the output should be printed to standard out (`System.out`).

General constraints and hints:

- Don't start from scratch, we provide you with a code skeleton to build upon. Feel free to improve or change the existing code, unless it is marked with "`DO NOT CHANGE`".
- About command line arguments:
    - Arguments in pointy brackets (<>) are mandatory, whereas those in square brackets are optional.
    - You can assume that arguments are always provided in the above order and with correct syntax.
- The order of lines from the input file must not be changed, i.e., the i-th row in the output corresponds to the (un-processed) i-th row of the input. In addition the output has to contains / print the basic statistics calculated using the provided `Statistics.java`
- Make sure that a small framework to easily add new operations and types is in place.
- We will also look at unit tests, so please add some. It's perfectly OK if you test only one specific part of the code -- if you prove you understand the basics of testing and code coverage.
- We will also look at Javadoc. Much like unit tests, you should write some, but it does not have to be complete -- if you prove you understand the basics of writing Javadoc.
- The coding challenge is split into three tasks (see below). The first task is mandatory, whereas the 2nd and 3rd are optional. The 2nd and 3rd task do not depend on each other.

Submission format:

Please send your submission via email prior to the interview (either as file attachment or as pointer to, e.g., github repo). Please submit only a single solution containing all tasks that you worked on.

In case you were not able to solve the 2nd and/or 3rd task (**which is not a problem at all!)**, yet you worked on them please also provide us with what you tried there. If you were not able to proceed with the task because you did not know, e.g., how to split your file into consecutive chunks simply create an empty method that would do that job and continue working on the task.

Task 1: Read - Transform - Write

Preliminaries

For this task you can assume that `--threads` == 1 and that only `--inputtype` == string

Description:

Write a program that reads your file, transforms each line to the required type and carries out the (chained) operations before writing the result back to the file / standard out. Make sure to implement the string related operations described above and don't forget to include the statistics calculated by the `Statistics.java` in your output.

Note:

Even if you decide not to work on Task 2 design the solution of this task in such a way that it would be easy to add support for additional input types and operations.

Task 2 (Optional): Support double and int input types

Preliminaries

For this task you can assume that `--threads` == 1

Description

Extend your code such that operations can also be carried out for integer and double input types (note that operations don't change the datatype) e.g.,

```
--input Input.txt
--inputtype int
--operations neg,rev
--threads 1
--output Output.txt
```

creates the following result:

Input.txt	Output.txt
102	-201
10	-1
-5	5

Task 3 (Optional): Multi-threading

Description

Extend your code such that the input file is read, parsed and transformed by concurrent threads. Each thread must process a different consecutive chunk of the input file. The chunks may only overlap in the beginning and end. Create only one chunk per thread.

Example for `--threads` == 3:

Input.txt	Thread ID
10	1
13	1
94	2
16	2
42	3
67	3

Nice to have

Once any of those threads fails with an exception all other threads need to be stopped and Main should throw a proper exception that describes the problem.