The purpose of this assignment is to learn how to use Streams in Java.

Exercise 1 (acquire the expertise) – Population

Import the source code contained in *es1* in your IDE. It's a slightly rearranged version of the first exercise of the previous assignment, which creates a list of random people (*Student, Worker,* and *Person*) and outputs statistics about the population. The computation of this statistics relies on anonymous and local nested classes.

The goal of this exercise is to remove all explicit 'for' loops and all the anonymous/nested classes from the *main()* method by replacing them with stream operations using lambda functions. At the end of the exercise the support interfaces *CategorizeOperation* and *EvaluateOperation* shouldn't be used anymore.

Proceed by following these suggestions:
1) Eliminate all 'for' loops by obtaining a stream from the *population* list and then operate on the stream.
2) Replace all implementations of:
   a) *EvaluateOperation* interface by moving their logic to the *filter()* stream operation. The *EvaluateOperation* interface is similar to the *Predicate* functional interface used by the *filter()* method.
   b) *CategorizeOperation* interface by using *collect()* in combination with *Collectors.groupingBy()* method. The *CategorizeOperation* interface is similar to the *Function* interface used by the *Collectors.groupingBy* method.
3) Use the *collect()* in combination with *Collectors.toList()* methods to aggregate results into a new list of *Person* instances.
4) When printing to console:
   a) Lists: obtain a stream from the desired list and use *map(), collect()* and *Collectors.joining()* methods to generate the string to be printed. The *map()* operation should be used to transform a *Person* instance to a *String*, using the *toString()* method.
   b) Maps: retrieve a stream of *Entry* from the given map's *entrySet()* and apply the *forEach()* method on the stream.

Exercise 2 (problem solving) – Text analyzer

Write a program that reads a text file and outputs information about the semantical structure of the text by using lambdas and streams. You are not allowed to use any for/while loops.

Start by reading the contents of a text file into a list of words by using the following code:
```
    Arrays.asList(new String(Files.readAllBytes(Paths.get("/path/to/file.txt"))).split("\\s+"))
```

Next, clean-up the list and generate a new list of strings containing only alphanumerical characters (both uppercase and lowercase). You can achieve this by removing all non-alphanumerical characters by using the following code snippet: `word.replaceAll("[^a-zA-Z0-9]", "")`. Then, exclude all empty strings from the resulting list.

Finally, use the newly created list of strings to find:
- the number of words with even length
- the list of words that start with a vowel
- the list of words that start with a capital T, without duplicates
- the list of words, without duplicates, by their length (1: ["a", "b"] 2: ["aa", "bb"], …)
- the number of words that start with a vowel and the number of words that do not start with a vowel
  Hint: create 2 groups: one representing the words that start with a vowel and the other group with the remaining words
- the number of each character (e.g. 'a': 132, 'b': 323, '3': 22, …)
  Hint: consider converting every word into a stream of individual characters. The *String.chars()*

method returns an *IntStream* containing the Unicode code points of each character, that can be converted to char using the casting operator.
- general statistics about the word lengths such as: sum of lengths, min, average and max length.

Exercise 3 (optional deepening) – Generators

Write a program that takes advantage of lambdas and streams to generates the following collections:

- All even numbers: this has to be done by using both *IntStream.iterate()* and *Stream.iterate()* methods.
- Playing card symbols (A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K): develop the solution by using both:
  - *Stream.of()* method
  - *IntStream.closedRange()* method in combination with a *map()* function that converts the int value to the corresponding char/String representing the card's symbol.
- An 18-character long password by generating a stream of random numbers in the ASCII range (40 - 120) and converting them to chars / Strings.
  Finally create the password *String* from the character stream by implementing 2 alternatives:
  - The first version must use *Collectors.joining(),*
  - The second alternative has to be implemented using *Stream.reduce().*
- The Fibonacci sequence (0, 1, 1, 2, 3, 5, 8, …) using the *LongStream.generate()* method.
  Hint: create a helper class to store two values (n1, n2) used to compute the sequence and a *next()* method. This *next()* method should return the current value and update n1 and n2 to compute the next Fibonacci value. Use the implemented *next()* method in combination with the *generate()* method.
  $F_n = F_{n-1} + F_{n-2}$;

  $F_0 = 0$
  $F_1 = 1$
  $F_2 = F_1 + F_0 = 1 + 0 = 1$
  $F_3 = F_2 + F_1 = 1 + 1 = 2$
  $F_4 = F_3 + F_2 = 2 + 1 = 3$
  $F_5 = F_4 + F_3 = 3 + 2 = 5$

Note: To evaluate the functionality of infinite sequences (like the even numbers or the Fibonacci sequence), process the stream up to a specific element count (using the *limit()* method) and consume the stream by displaying the contents on the console.