Stream Processing in Java 8

Introduction

For each of the following exercises provide two solutions. The first solution must use the standard iteration constructs; the second solution must use methods of the Stream interface interface. Futhermore, use Java generics whenever possible.

Exercise 1

Write a static method `sumOdd` that given a `List` of integers computes the sum of the values that are *odd*.

Goal: Warming up!

Expected output: A properly commented Java source file containing the two requested methods.

Exercise 2

Implement a generic class `ImmutablePair<T1,T2>`. Besides a suitable constructor your class should provide also methods to get each element individually. Then, write a static method `someCalculation` (**not** in `ImmutablePair<T1,T2>`) that given a `List<Double> lst` returns an object of `ImmutablePair<Integer,Double>` class. The *first* element of the pair is the number of elements of `lst` in the range `[0.2, Math.PI]`, and the *second* element is the average of the values of `lst` in the range `[10, 100]`.

Goal: Experimenting with the basics of Java's Stream API.

Expected output: Properly commented Java files for the class and for the requested methods.

Exercise 3

Write a static method `repl` that given an array `xs` of `Object` and a integer `n` returns an array containing the elements of `xs` replicated `n` times in any order.

Hint: For the stream-based version, consider the `flatMap` method of `Stream`.

Question: What happens if the parameter `xs` has type `List<T>`? How do you need to change `repl` to return an array of `T` in this case?

Goal: Exploring the Stream API further and its relation with types in Java.

Expected output: A properly commented Java source file implementing the requested method.

Exercise 4

Write a `static` method `titlecase` that given a string `s` converts it to *titlecase* by upper-casing the first letter of every word.

Goal: Re-implementing an old Haskell exercise in Java, comparing the solutions.

Expected output: A properly commented Java source file implementing the requested method.

Exercise 5

Write a static method `zipWithIndex` that given an array of type `T` returns a `Stream<ImmutablePair<T, Integer>>` containing the elements of the array with its indexes. Use the method just defined to implement another static method `filterOdd` that given an array `xs` returns a new array obtained from `xs` by removing the elements at odd positions.

Hint: Here "odd positions" means the first, third, fifth, etc position.

Goal: Re-implementing an old Haskell exercise in Java, comparing the solutions (pt. II).

Expected output: A properly commented Java source file implementing the requested method.

Exercise 6

Write a static method `replaceWord` that given the name of a text file (`String fileName`), a word (`String word`) and a replacement (`String repl`), prints the content of `fileName` where each occurrence of `word` is replaced with `repl`. Note that the structure, i.e. lines, of `fileName` must be preserved. Test your method with the file `people.csv`, replacing for example `true` with `false`.

Hint: Consider the method Files.lines of Java API.

Goal: Working with more advanced aspects of Java's Stream API.

Expected output: A properly commented Java source file implementing the requested method.

Exercise 7

Implement a static method `serialEvenSum` that given a `long threshold` as parameter computes the sum of even numbers up to the threshold using the Stream API. Futhermore, write a static method `parallelEvenSum` that does the same but using a parallel stream. Finally, write another static method `testSum` that takes a `long threshold` as argument, runs the two methods just defined, measures and compares their running time. Invoke `testSum` for increasing values of the `threshold` and determine for which values of `threshold` using serial streams is better than using the parallel ones and vice versa.

Hint: Consider the method System.nanoTime to measure time.

Goal: Working with more advanced aspects (parallelism) of Java's Stream API.

Expected output: A properly commented Java source file implementing the requested method.

Exercise 8

Write a method `getElement` implementing a *partial function* that given an array `arr` and an `int index` returns `arr[index]` if defined. Use Optional as the result type. Next write methods implementing the following partial functions:

- `sqrt`, that applied to an integer returns its square root as a double, if the argument is not negative;
- `half`, that applied to an integer returns its half as an integer, if the argument is even.

Finally write a method that composing `getElement`, `sqrt` and `half`, when applied to an array of integers `arr` and to an `int index`, returns the square root of the half of `arr[index]`, if defined.

Goal: Working with partial functions in Java, exploiting the `Optional` class.

Expected output: A properly commented Java source file implementing the requested methods.

Exercise 9

Write a static method `listDir` that given the path of a directory as argument returns a `List<String>` containing the names of all the subdirectories recursively contained in it.

Hint: Consider to use the methods of the Files class, e.g. `list` or `walk`.

Goal: Accessing system's informations though Java.

Expected output: A properly commented Java source file implementing the requested method.

Exercise 10

Consider the attached csv file `people.csv`. This file stores information about people subscribed to a simple service. Each line of the file represents a record. The fields of each record are separated by a comma "," and have the following meaning

```
id,firstname,surname,title,address,town,country,postcode,subscription
paid,gender,date of birth
```

In a record the fields `id`, `firstname`, `surname`, `date of birth`, `subscription paid` are mandatory, while the other are optional (denoted by "-" in the file). Implement a class `Subscriber` representing a subscriber, providing a property for each field. Use the Optional class to denote optional fields.

Write a static method `loadDatabase` that returns a `List<Subscriber>` containing the records of the file `subscriber.csv`. Furthermore, implement another static method `PaymentFromGB` that given a `List<Subscriber>` prints the subscribers from GB that have paid the annual fee.

Hint: Use the methods of Optional class to deal with optional values.

Goal: Working with more advanced aspects of Java's Stream API.

Expected output: A properly commented Java source file implementing the requested method.

Author: Andrea Corradini & Matteo Busi

Created: 2018-11-25 dom 23:53

Validate