

Exercises week 10

Last update: 2024/10/30

These exercises aim to give you practical experience with Java streams and recognize the key concepts of RxJava in particular Observer/Observable.

- Enable you to use Java streams and parallel streams.
- Enable you to parallelize Java streams.

Exercise 10.1 Consider this program that computes prime numbers using a while loop:

```
class PrimeCountingPerf {
    public static void main(String[] args) { new PrimeCountingPerf(); }
    static final int range= 100000;
    //Test whether n is a prime number
    private static boolean isPrime(int n) {
        int k= 2;
        while (k * k <= n && n % k != 0)
            k++;
        return n >= 2 && k * k > n;
    }
    // Sequential solution
    private static long countSequential(int range) {
        long count = 0;
        final int from = 0, to = range;
        for (int i=from; i<to; i++)
            if (isPrime(i)) count++;
        return count;
    }
    // IntStream solution
    private static long countIntStream(int range) {
        long count= 0;
        // to be filled out
        return count;
    }
    // Parallel Stream solution
    private static long countParallel(int range) {
        long count= 0;
        // to be filled out
        return count;
    }
    // parallelStream solution
    private static long countparallelStream(List<Integer> list) {
        long count= 0;
        // to be filled out
        return count;
    }
    public PrimeCountingPerf() {
        Benchmark.Mark7("Sequential", i -> countSequential(range));
        Benchmark.Mark7("IntStream", i -> countIntStream(range));
        Benchmark.Mark7("Parallel", i -> countParallel(range));
        List<Integer> list = new ArrayList<Integer>();
        for (int i= 2; i< range; i++){ list.add(i); }
        Benchmark.Mark7("ParallelStream", i -> countparallelStream(list));
    }
}
```

You may find this in `Week10/code-exercises ... /PrimeCountingPerf.java`. In addition to counting the number of primes (in the range: `2..range`) this program also measures the running time of the loop. Note, in your solution you may change this declaration (and initialization) `long count= 0;`

1. Compile and run `PrimeCountingPerf.java`. Record the result in a text file.
2. Fill in the Java code using a stream for counting the number of primes (in the range: `2..range`). Record the result in a text file.
3. Add code to the stream expression that prints all the primes in the range `2..range`.
4. Fill in the Java code using the intermediate operation `parallel` for counting the number of primes (in the range: `2..range`). Record the result in a text file.
5. Add another prime counting method using a `parallelStream` for counting the number of primes (in the range: `2..range`). Measure its performance using `Mark7` in a way similar to how we measured the performance of the other three ways of counting primes.

Exercise 10.2 This exercise is about processing a large body of English words, using streams of strings. In particular, we use the words in the file `app/src/main/resources/english-words.txt`, in the exercises project directory.

The exercises below should be solved without any explicit loops (or recursion) as far as possible (that is, use streams).

1. Starting from the `TestWordStream.java` file, complete the `readWords` method and check that you can read the file as a stream and count the number of English words in it. For the `english-words.txt` file on the course homepage the result should be 235,886.
2. Write a stream pipeline to print the first 100 words from the file.
3. Write a stream pipeline to find and print all words that have at least 22 letters.
4. Write a stream pipeline to find and print some word that has at least 22 letters.
5. Write a method `boolean isPalindrome(String s)` that tests whether a word `s` is a palindrome: a word that is the same spelled forward and backward. Write a stream pipeline to find all palindromes and print them.
6. Make a parallel version of the palindrome-printing stream pipeline. Is it possible to observe whether it is faster or slower than the sequential one?
7. Make a new version of the method `readWordStream` which can fetch the list of words from the internet. There is a (slightly modified) version of the word list at this URL:
<https://staunstrup.dk/jst/english-words.txt>. Use this version of `readWordStream` to count the number of words (similarly to question 7.2.1). Note, the number of words is *not* the same in the two files !!
8. Use a stream pipeline that turns the stream of words into a stream of their lengths, to find and print the minimal, maximal and average word lengths.
Hint: There is a simple solution using an operator exemplified on p. 141 of *Java Precisely* (included in the readings for this week).

Exercise 10.3 This exercise is based on the article: (Introduction to Java 8 Parallel Stream) (on the readme for week10). Start by reading this.

1. Redo the first example (running the code in `Java8ParallelStreamMain`) described in the article. Your solution should contain the output from doing this experiment and a short explanation of your output.
2. Increase the size of the integer array (from the 10 in the article) and see if there is a relation between number of cores on your computer and the number of workers in the `ForkJoin`.
3. Change the example by adding a time consuming task (e.g. counting primes in a limited range or the example in the article). Report what you see when running the example.

Exercise 10.4 Despite many superficial syntactical similarities between `JavaStream` and `RxJava`, the two concepts are fundamentally different. This exercise focus on some of these differences.

Consider the pseudo-code below (that does not compile and run). The `source()` provides english words and the `sink()` absorbs them. Note that `sink()` is a pseudo-operation that just absorbs the data it receives. Your explanations should focus on what happens in between the `source()` and `sink()`.

1. Describe what happens when this code runs:

```
source().filter(w -> w.length() > 5).sink()
```

- as a `JavaStream` (e.g. the source is a file)
- as a `RxJava` statement where the source could be an input field where a user types strings

2. Describe what happens when this code runs:

```
source().filter(w -> w.length() > 5).sink();  
source().filter(w -> w.length() > 10).sink()
```

- as a `JavaStream` (e.g. the source is a file)
- as a `RxJava` statement where the source could be an input field where a user types strings

The solution to this exercise is just a short explanation in English - there is no code to develop and run.