Worksheet: Streams

Lab Exercises

You should start by importing the appropriate project into your IDE. Project files are provided for Eclipse and IntelliJ on the repo. All the following questions should be answered using the new Java 8 *Streams*.

For all the following exercises, start with a `List` of `String`s similar to this:

```
List<String> words = Arrays.asList("hi", "hello", ...);
```

1. Loop through the words and print each on a separate line, with two spaces in front of each word.
2. Repeat the previous problem, but without the two spaces in front. This is trivial if you use the same approach as in (1), so the point here is to use a method reference.
3. For the following lists produce the same transformations using `map`:
   - `List<String> excitingWords = StringUtils.transformedList(words, s -> s + "!");`
   - `List<String> eyeWords = StringUtils.transformedList(words, s -> s.replace("i", "eye"));`
   - `List<String> upperCaseWords = StringUtils.transformedList(words, String::toUpperCase);`
4. For the following lists produce the same transformations using `filter`:
   - `List<String> shortWords = StringUtils.allMatches(words, s -> s.length() < 4);`
   - `List<String> wordsWithB = StringUtils.allMatches(words, s -> s.contains("b"));`
   - `List<String> evenLengthWords = StringUtils.allMatches(words, s -> (s.length() % 2) == 0);`
5. Turn the strings in the array `words` into uppercase, keep only the ones that are shorter than 4 characters, and, of what is remaining, keep only the ones that contain `"e"`, and print the first result. Repeat the process, except checking for a `"q"` instead of an `"e"`.
6. The above example uses lazy evaluation, but it is not easy to see that it is doing so. Make a variation of the above example that proves that it is doing lazy evaluation. The simplest way is to track which entries are turned into upper case.

7. Produce a single `String` that is the result of concatenating the uppercase versions of all of the `String`s. E.g., the result should be `"HIHELLO..."`. Use a single `reduce` operation, without using `map`.

8. Produce the same `String` as above, but this time via a `map` operation that turns the words into upper case, followed by a `reduce` operation that concatenates them.

9. Produce a `String` that is all the words concatenated together, but with commas in between. E.g., the result should be `"hi,hello,..."`. Note that there is no comma at the beginning, before `"hi"`, and also no comma at the end, after the last word.

10. Write a `static` method that produces a `List` of a specified length of random numbers. E.g.:

```
List<Double> nums = StreamUtils.randomNumberList(someSize);

// Result is something like [0.7096867136897776, 0.09894202723079482, ...]
```

11. Write a `static` method that produces a list of numbers that go in order by a step size. E.g.:

```
List<Integer> nums = StreamUtils.orderedNumberList(50, 5, someSize);

// Result is [50, 55, 60, ...]
```

12. The lecture slides illustrated three ways to use streams to compute sums. Use one of the three variations, and compute the sum of some `int`s.

13. Rewrite the solution for (12) so that it can be executed in parallel; verify that you get the same answer both times.

14. Now, use streams to compute the product of some doubles. Show that serial and parallel versions **do not** always give the same answer.

    (Note: this is a bit tricky, because it seems at first that multiplication is associative, as required by the parallel `reduce`. It will be impossible to illustrate the result if you have a single-core computer.)