Kevin Lim
lyskevin@u.nus.edu

Fabian Chia
fabianc@u.nus.edu

CS2030 Lab 7

AY19/20 Sem 2, Week 9

Midterms

Your code has been uploaded onto LumiNUS

You have until this Sunday (22 March 2020) to modify your solutions to pass all test cases (not necessary if your solution is already correct)

You must still submit a working solution (that passes all test cases) even if you did not attempt the question

Marks will be awarded based on how much you modify your code (full marks if your code remains unmodified)

Project Stage 2

Stage 2 of the project has been released on CodeCrunch

Solutions must be submitted by next Sunday (29 March 2020)

Again, plan properly before starting to code

Lab 7 – Streams – Introduction

A stream is an infinite sequence of elements

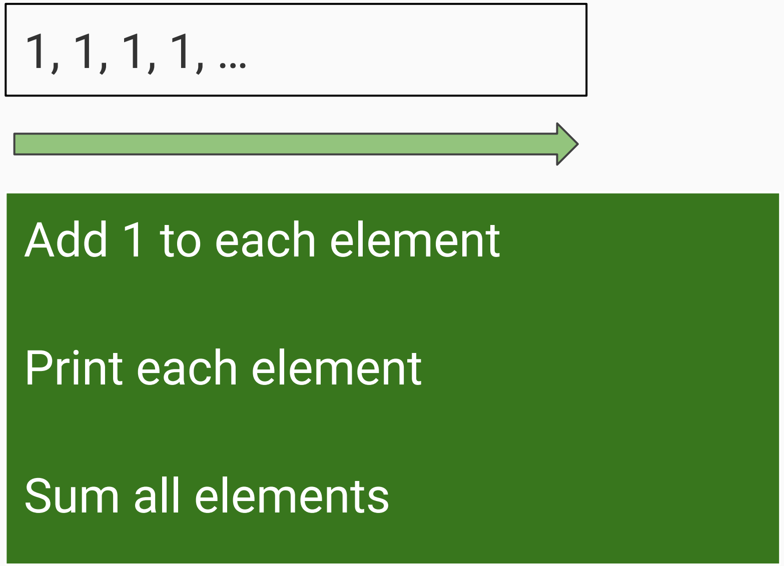
Example: A stream of 1s

1, 1, 1, 1, …

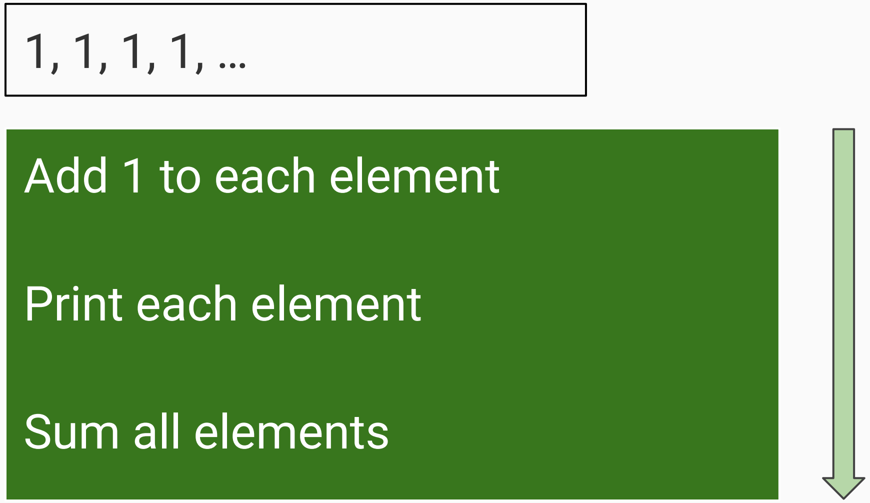
Different kinds of streams are supported by the Java API: Stream (Most general), IntStream (Stream of ints), DoubleStream (Stream of doubles), etc.

Lab 7 – Streams – Lazy evaluation

Stream elements are lazily evaluated. Consider the following stream of 1s. The stream on the left is lazily evaluated and the one on the right is eagerly evaluated.



Lazy evaluation: Perform each operation on ALL elements before moving to the next operation



Eager evaluation: Perform ALL operations on each element before moving to the next element (e.g. multiple operations inside a for loop)

Lab 7 – Streams – Operation types

There are 2 main kinds of operations: intermediate and terminal operations.

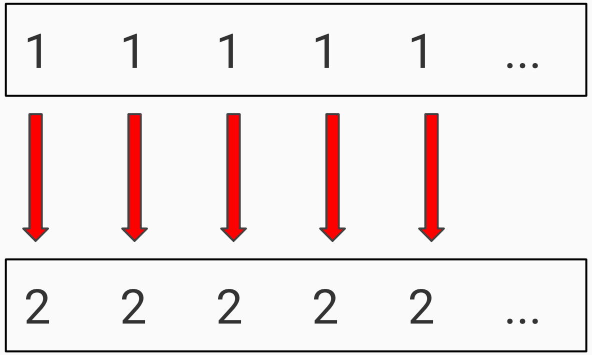
Intermediate operations produce another stream (e.g. map, flatMap, filter, etc.).

Terminal operations return a **result** and stream operations can no longer be performed after a terminal operation. Examples of terminal operations include count and reduce.

Stream operations can be called in a sequential manner using method chaining.

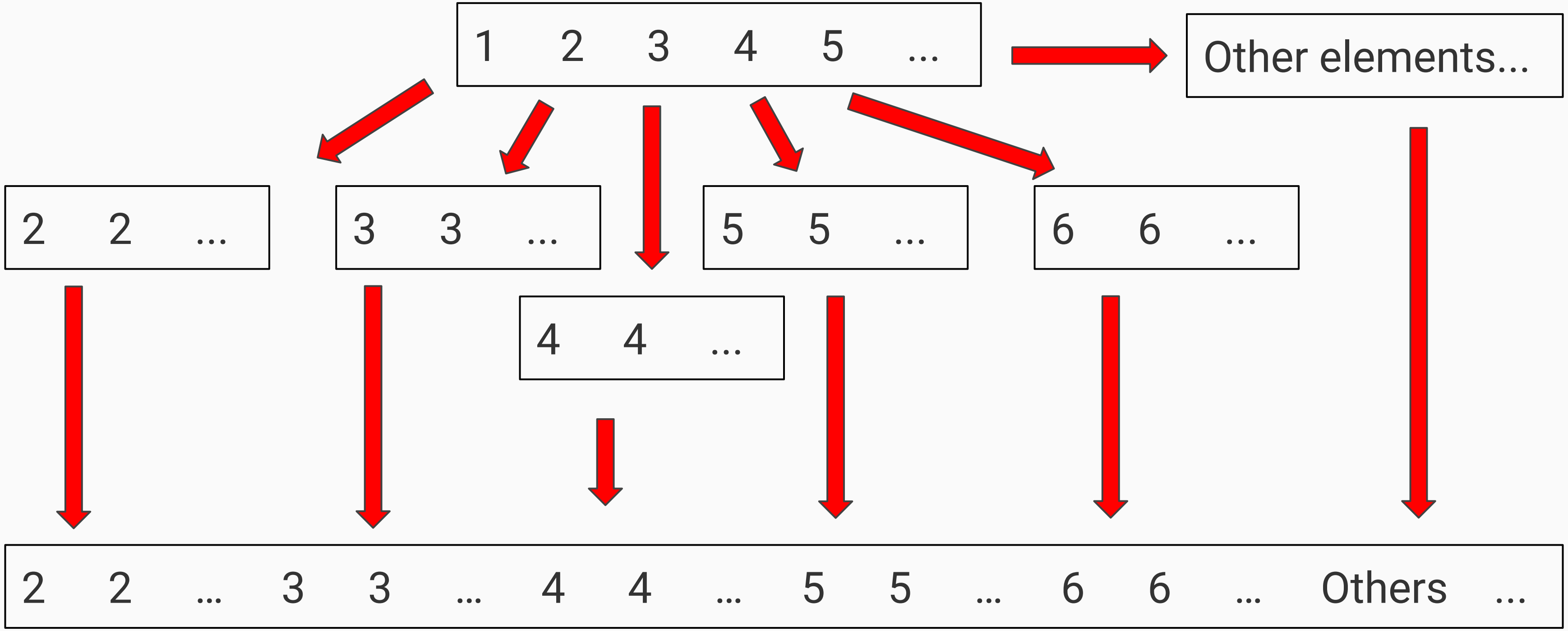
Lab 7 – Streams – MAP VS FLATMAP (again)

map: Maps each stream element to another element (could be same type)



Lab 7 – Streams – MAP VS FLATMAP (again)

flatMap: Maps **each** stream element to a **stream of elements,** then **flattens all streams** together.



Lab 7 – Streams – Stuck?

If you’re not sure where to begin, write out the normal version of code without streams, then try to pick out parts which can be implemented with streams.

Eg. factors of a number X, from 2 to X.

Normal pseudocode:

1. Iterate through numbers 2 to X.
2. If X % currNumber == 0, store the currNumber in a list.
3. By the end of the for loop, the list should contain all the factors of X.

Streams equivalent:

1. Create a stream of numbers 2 to X.
2. Go through the stream of numbers, and if X % currNumber == 0, keep it. (How? When in doubt, look at the API.)
3. By the end of the stream evaluation, the somewhere you have chosen should contain all the factors of X. ((((: