Stream exercises

Tutorial 12 (12th May 2021)

Ike Mulder

Radboud University

Solving the exercises

Given *data*, produce *result*

- Turn *data* into a Stream using a <u>source</u> method
- Use <u>intermediate operations</u> to get the stream you want
- Use a <u>terminal</u> method to turn this stream into *result*

Getting the types right (1)

Java notation of a function taking Strings to Integers:

```
  Function<? super String, ? extends Integer>
```

In this tutorial we use the notation:

String ⇒ Integer

Getting the types right (2)

In this tutorial we use the notation:

$A \Rightarrow B$

for functions sending objects of type A to objects of type B

- `(String s) -> Integer.parseInt(s)`
  - String ⇒ int
- `(int i) -> (i < 4)`
  - int ⇒ boolean
- `(double l, double r) -> d * r`
  - (double, double) ⇒ double

Getting the types right (2)

In this tutorial we use the notation:

$A \Rightarrow B$

for functions sending objects of type A to objects of type B

- `Integer::parseInt`
  - String ⇒ int

Stream building blocks: sources

Method	Type
`Arrays.stream`	T[ ] ⇒ Stream<T>
	int[ ] ⇒ IntStream
	double[ ] ⇒ DoubleStream

```
int[] e = {2,7,1,8,2,8};
IntStream result = Arrays.stream(e);
```

Stream building blocks: intermediate operations

```
Stream<T> stream;
```

Method	Type
`stream.filter`	(T ⇒ boolean) ⇒ Stream<T>
`stream.limit`	int ⇒ Stream<T>
`stream.sorted`	Comparator<T> ⇒ Stream<T>
`stream.map`	(T ⇒ U) ⇒ Stream<U>
`stream.mapToInt`	(T ⇒ int) ⇒ IntStream

Stream building blocks: terminal operations

```
Stream<T> stream;
IntStream intStream;
```

Method	Type
`stream.count`	int
`stream.reduce`	$(U, (U, T) \Rightarrow U) \Rightarrow U$
`intStream.sum()`	int

Stream exercises

Exercise	Difficulty	remarks
countEvenNumbers	⋆	
sumOddNumbers	⋆	
multiplyNumbers	⋆	
calculateTax	⋆	
sumStringIntegers	⋆	
streamOfStreams	⋆ ⋆ ⋆	flatMap
everySecondElement	⋆ ⋆	list collector
testRot13	⋆ ⋆ ⋆	custom collector
philosophers	⋆ ⋆	map collector

Stream building blocks: other sources

Method	Type
`Arrays.stream`	T[ ] ⇒ Stream<T>
	int[ ] ⇒ IntStream
	double[ ] ⇒ DoubleStream

```
Collection<T> col; String string;

Stream<T> colStream = col.stream();
// works for List<T>, Set<T>, etc

IntStream codepointsStream = string.codePoints();
```

Stream building blocks: intermediate operations - flatMap

`Stream<T> stream;`

Method	Type
`stream.flatMap`	(T ⇒ Stream<U>) ⇒ Stream<U>
`stream.flatMapToInt`	(T ⇒ IntStream) ⇒ IntStream

for T = Stream<U>, flatMap ‘flattens’,

turns a `Stream<Stream<T>>` into one big `Stream<T>`

by concatenating all streams

Stream building blocks: terminal operations - collect

```
Stream<T> stream;
```

Method	Type
`stream.collect`	Collector<T, A, U> ⇒ U

Creates (empty) result object

Consumes stream element

Combines result objects

(Void ⇒ U,
(U , T) ⇒ Void,
(U , U) ⇒ U) ⇒ U

Finally

Radboud University