USF UNIVERSITY OF SAN FRANCISCO

CHANGE THE WORLD FROM HERE

Stream Basics

CS 272 Software Development

Department of Computer Science
https://www.cs.usfca.edu/

Java Streams

- Added alongside **lambda expressions** in Java SE 8
- Allows **functional-style** operations
  - i.e. map-reduce transformations
    (not to be confused with MapReduce)
- Allows **function composition**
  - i.e. combining simple functions into complex ones

Project Euler Problem 1

Find the sum of all the
multiples of 3 or 5 below 1000.

https://projecteuler.net/problem=1

Traditional Approach

```
1. int max = 1000;
2. int sum = 0;
3.
4. for (int i = 0; i < max; i++) {
5.   if (i % 3 == 0 || i % 5 == 0) {
6.     sum += i;
7.   }
8. }
```

https://projecteuler.net/problem=1

Traditional Approach

```
1. int max = 1000;
2. int sum = 0;
3.
4. for (int i = 0; i < max; i++) {
5.   if (i % 3 == 0 || i % 5 == 0) {
6.     sum += i;
7.   }
8. }
```

1. Generate data.

https://projecteuler.net/problem=1

Traditional Approach

```
1. int max = 1000;
2. int sum = 0;
3.
4. for (int i = 0; i < max; i++) {
5.   if (i % 3 == 0 || i % 5 == 0) {
6.     sum += i;
7.   }
8. }
```

2. Filter data.

https://projecteuler.net/problem=1

Traditional Approach

```
1. int max = 1000;
2. int sum = 0;
3.
4. for (int i = 0; i < max; i++) {
5.   if (i % 3 == 0 || i % 5 == 0) {
6.     sum += i;
7.   }
8. }
```

3. Reduce data.

https://projecteuler.net/problem=1

Functional Approach

```
1. int max = 1000;
2.
3. int sum = IntStream.range(0, max)
4.    .filter(i → i % 3 == 0 || i % 5 == 0)
5.    .sum();
```

https://projecteuler.net/problem=1

Functional Approach

```
1. int max = 1000;
2.
3. int sum = IntStream.range(0, max)
4.    .filter(i → i % 3 == 0 || i % 5 == 0)
5.    .sum();
```

1. Generate data.

https://projecteuler.net/problem=1

Functional Approach

```
1. int max = 1000;
2.
3. int sum = IntStream.range(0, max)
4.    .filter(i → i % 3 == 0 || i % 5 == 0)
5.    .sum();
```

2. Filter data.

https://projecteuler.net/problem=1

Functional Approach

```
1. int max = 1000;
2.
3. int sum = IntStream.range(0, max)
4.    .filter(i → i % 3 == 0 || i % 5 == 0)
5.    .sum();
```

3. Reduce data.

https://projecteuler.net/problem=1

Traditional vs Functional

```
1. for (int i = 0; i < max; i++)
2.   if (i % 3 == 0 || i % 5 == 0)
3.     sum += i;
4.
5. IntStream.range(0, max)
6.   .filter(i → i % 3 == 0 || i % 5 == 0)
7.   .sum();
```

https://projecteuler.net/problem=1

Stream	Collection
Computation Pipeline (Actions)	Data Structure (Storage)
Infinite or Finite Data Size	Finite Data Size
Lazy Intermediate Operations *e.g. performed when a result is needed*	Eager Operations *e.g. performed immediately*
Functional Operations *e.g. does not modify the source data*	Mutative Operations *e.g. modifies the source data*
Consumable *e.g. can only be used once*	Persistent *e.g. can be reused multiple times*

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/util/stream/package-summary.html

Data Source	Description
`Collection.stream()`	Stream of elements in a collection.
`Stream.of(T[])`	Stream of an array.
`BufferedReader.lines()`	Stream of lines in a file.
`CharSequence.chars()`	Stream of characters in text.
`IntStream.range( ... )`	Stream of numbers within a range.
`Stream.iterate( ... )` `Stream.generate( ... )`	Infinite stream of elements created via iteration or a generator function.

https://developer.ibm.com/articles/j-java-streams-1-brian-goetz/

References

Package java.util.stream
https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/util/stream/package-summary.html

The Java Tutorials – Lesson: Aggregate Operations
https://docs.oracle.com/javase/tutorial/collections/streams/index.html

“An introduction to the java.util.stream library” by Brian Goetz
https://developer.ibm.com/articles/j-java-streams-1-brian-goetz/

CHANGE THE WORLD FROM HERE

Professor Sophie Engle
sjengle.cs.usfca.edu