ORACLE®

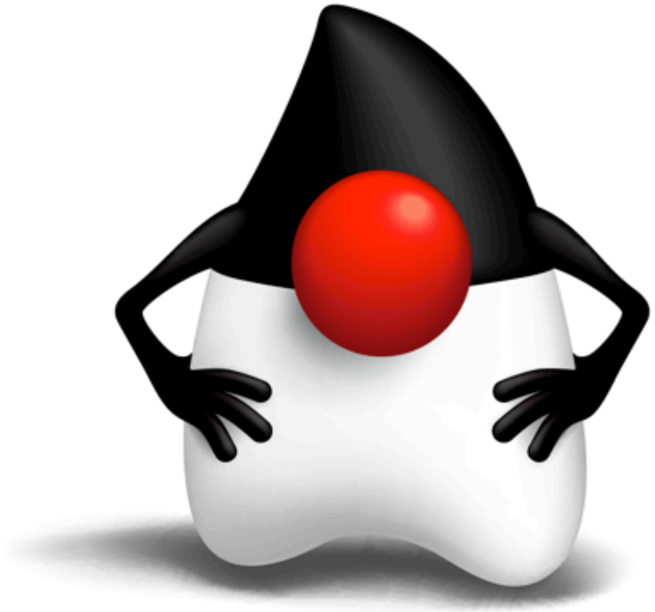
ambdas & Streams Laboratory

CREATE
THE
FUTURE

mon Ritter
racle Corp.

witter: @speakjava

Lambdas and Functions Library Review



ambda Expressions

Lambda expression is an anonymous function

Think of it like a method

– But not associated with a class

Can be used wherever you would use an anonymous inner class

– Single abstract method type

Syntax

– `( [optional-parameters] ) -> body`

Types can be inferred (parameters and return type)

ambda Examples

```
SomeList<Student> students = ...
double highestScore =
  students.stream().
           filter(Student s -> s.getGradYear() == 2011).
           map(Student s -> s.getScore()).
           max();
```

Method References

Method references let us reuse a method as a lambda expression

```
FileFilter x = (File f) -> f.canRead();
```

```
FileFilter x = File::canRead;
```

he Stream Class

va.util.stream

`Stream<T>`

– A sequence of elements supporting sequential and parallel operations

A Stream is opened by calling:

– `Collection.stream()`

– `Collection.parallelStream()`

Many Stream methods return Stream objects

– Very simple (and logical) method chaining

tream Basics

Using a Stream means having three things

A source

– Something that creates a `Stream` of objects

Zero or more intermediate objects

– Take a `Stream` as input, produce a `Stream` as output

– Potentially modify the contents of the `Stream` (but don’t have to)

A terminal operation

– Takes a `Stream` as input

– Consumes the `Stream`, or generates some other type of output

tream Usage

- Multiple operations available
  - `collect`, `filter`, `count`, `skip`, `limit`, `sorted`
  - `map` (and map to types, e.g. `mapToInt`)
  - `flatMap` maps each element in a `Stream` to possibly multiple elements
    - e.g. `flatMap(line -> Stream.of(line.split(REGEXP));`

```
List<String> names = Arrays.asList(“Bob”, “Alice”, “Charlie”);
System.out.println(names.
  stream().
  filter(e -> e.getLength() > 4).
  findFirst().
  get());
```

ava.util.function Package

`Predicate<T>`

– Determine if the input of type T matches some criteria

`Consumer<T>`

– Accept a single input argumentof type T, and return no result

`Function<T, R>`

– Apply a function to the input type T, generating a result of type R

`Supplier<T>`

– A supplier of results of type T

Plus several more type specific versions

he iterable Interface

sed by most collections

One method
- forEach()
- The parameter is a Consumer

```
wordList.forEach(s -> System.out.println(s));

wordList.forEach(System.out::println);
```

iles and Lines of Text

BufferedReader has new method

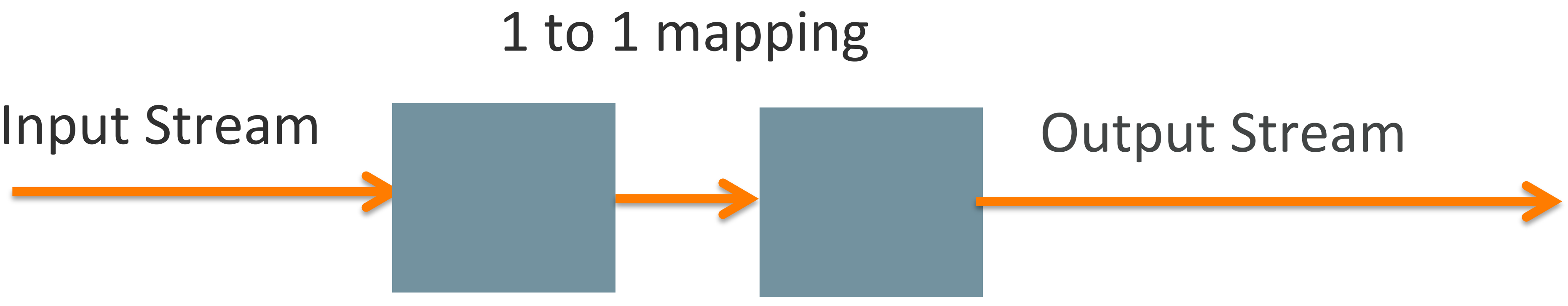
– `Stream<String> lines()`

HINT: Test framework creates a `BufferedReader` for you

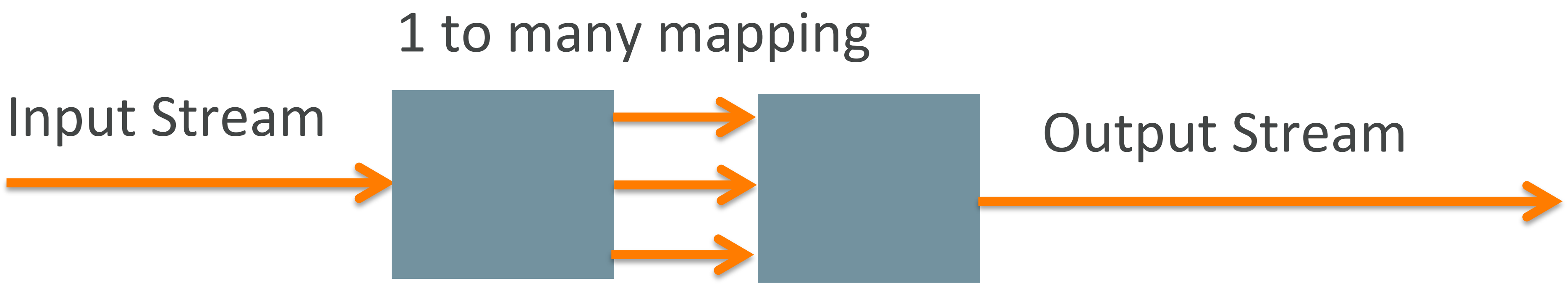
laps and FlatMaps

lap Values in a Stream

Map



tMap



Jseful Stream Methods

- collect (terminal)
- filter (intermediate)
- count (terminal)
- skip, limit (intermediate)
- max (terminal)
- getAsInt (terminal)

etting Started

- Open the LambdasHOL project in Eclipse
- The exercises are configured as tests
- Edit each test's method
  - Remove the `@Ignore` annotation
- Run the tests
  - Right-click the `Exercises.java` file and select `Run as > JUnit Test`
- Make the tests pass
- Simple!

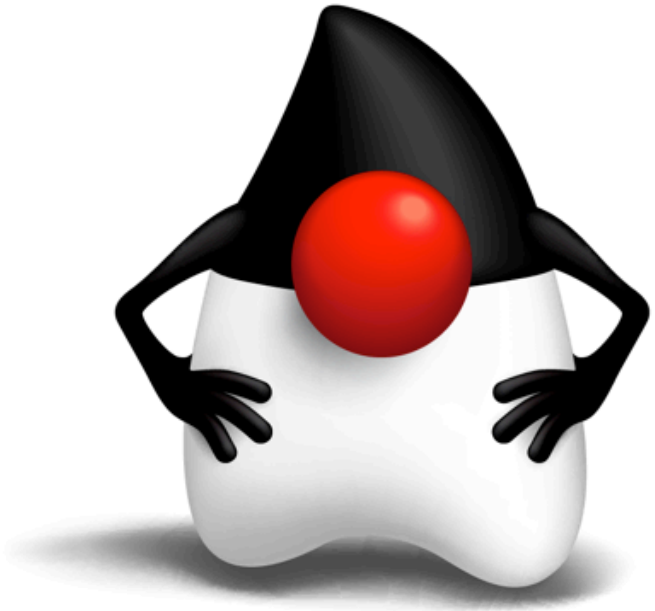
ccess To The Files

. USB keys at front

. www.github.com/speakjava/Lambda_Lab-EclipseCon

. Micro router (10.0.1.254)
  - ESSID: NANO_NOMIS
  - Workgroup: NOMIS

Let’s Go!



ORACLE

Java™