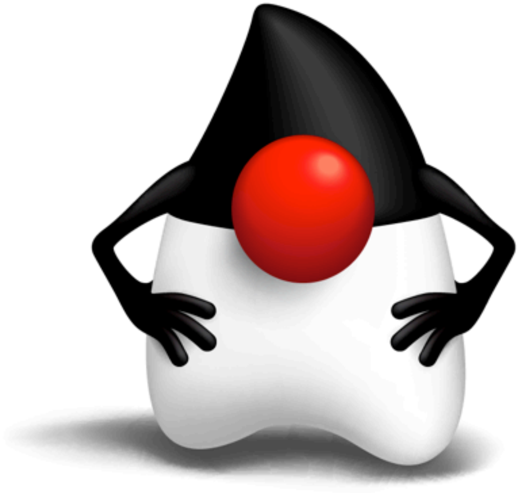
Lambdas & Streams Laboratory

CREATE
THE
FUTURE

Simon Ritter
Oracle Corp.

Twitter: @speakjava

Lambdas and Functions
Library Review



Lambda 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)

Lambda Examples

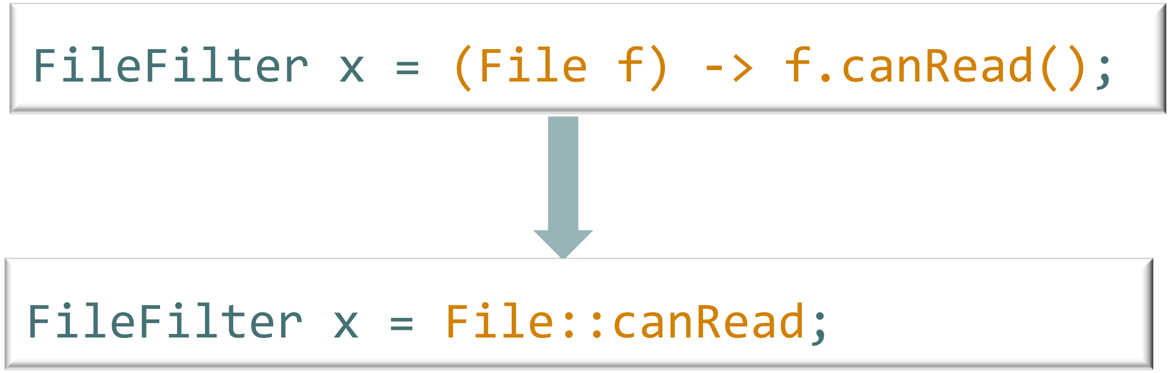
```
(a, b) -> a + b

s -> s.getGradYear() == 2011

() -> System.out.println("New Thread")
```

Method References

- Method references let us reuse a method as a lambda expression



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

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

Stream 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

The Stream Class

java.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

java.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
- `BiFunction<T, U, R>`
  - Apply a function that takes two arguments of type T and U, generating a result of type R
- `Supplier<T>`
  - A supplier of results of type T

The `iterable` Interface

Used by most collections

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

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

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

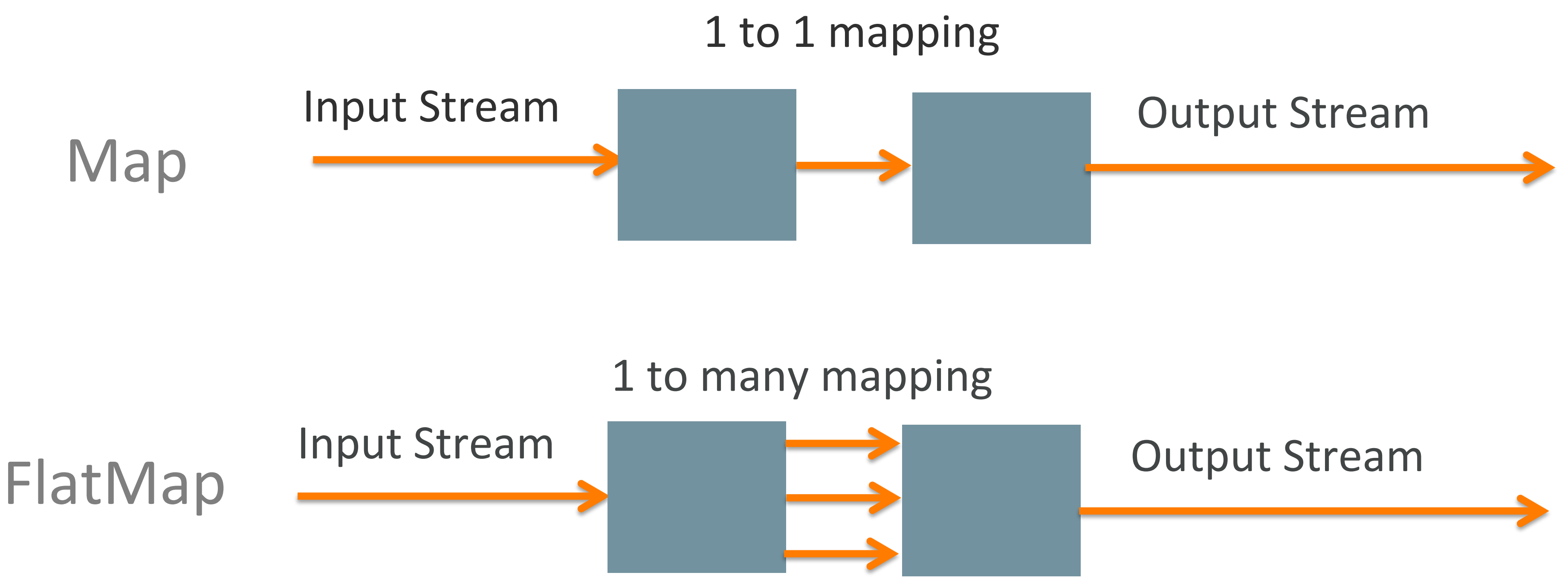
Files and Lines of Text

- BufferedReader has new method
  - `Stream<String> lines()`

- HINT: Test framework creates a `BufferedReader` for you

Maps and FlatMaps

Map Values in a Stream



Useful Stream Methods

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

Getting Started

- Make sure you have the required software installed:
  - JDK8 and documentation
  - NetBeans or Eclipse
- Open the appropriate project for your IDE

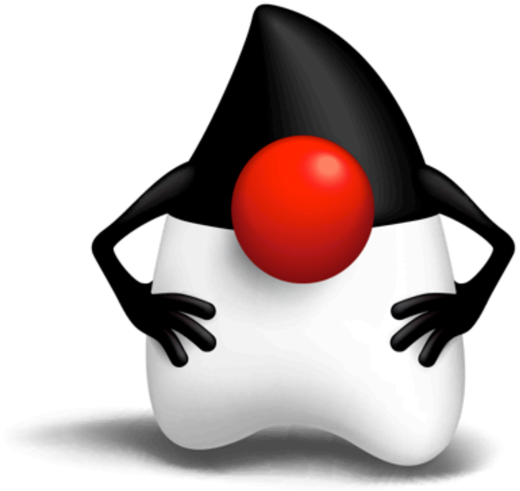
Working Through The Exercises

- Each exercise is a JUnit test
- Each exercise has it's own method
  - There is a comment to explain the goal of the exercise
  - There are comments to provide hints to help
- Edit the method and write the code
- Remove the `@Ignore` annotation above the method
- Run the tests
  - NetBeans: Run > Test Project (Or Shift F6)
  - Eclipse: Right click Exercises.java, Run As > JUnit Test
- All the solutions are provided if you get really stuck

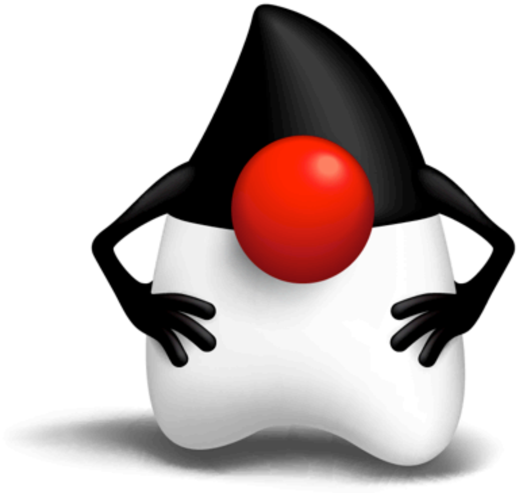
Access To The Files

1. USB keys at front
2. www.github.com/speakjava/Lambda_Lab-NetBeans
3. www.github.com/speakjava/Lambda_Lab-EclipseCon
4. Micro router (10.0.1.254)
   - ESSID: NANO_NOMIS
   - Workgroup: NOMIS

Let’s Go!



Solutions



Exercise 1

First character of each word concatenated

```
StringBuilder sb = new StringBuilder();
input.forEach(s -> s.append(s.charAt(0)));
String result = sb.toString();
```

```
String result = input.stream()
  .map(s -> s.substring(0, 1))
  .reduce("", (a, b) -> a + b);
```

Exercise 5

Map whose keys are first letter, values are sum of lengths

```
Map<String, Integer> result = new TreeMap();

list.forEach(s ->
  result.merge(s.substring(0, 1),
               s.length(),
               Integer::sum));
```

Exercise 9

Find the length of the longest line

```
int longest = reader.lines()
  .mapToInt(String::length)
  .max()
  .getAsInt();
```

Exercise 10

Find the longest line

```
String longest = reader.lines()
  .max(comparingInt(String::length))
  .get();
```

Exercise 11

Select the set of words whose length is greater than the word's position

```
List<String> result = IntStream.range(0, input.size())
  .filter(pos -> input.get(pos).length() > pos)
  .mpToObj(pos -> input.get(pos))
  .collect(toList());
```

Exercise 13

Convert a list of strings into a list of characters

```
List<Character> result = input.stream()
  .flatMap(word -> word.chars().mapToObj(i -> (char)i))
  .map(c -> (Character)c)
  ,collect(toList());
```

Eclipse only: bug

Exercise 17

Sort unique, lower-cased words by length, then alphabetically within length

```
List<String> result = reader.lines()
  .flatMap(line -> Stream.of(line.split(REGEXP)))
  .map(String::toLowerCase)
  .distinct()
  .sorted(comparingInt(String::length)
    .thenComparing(naturalOrder()))
  .collect(toList());
```

Exercise 18

Count total number of words and distinct lower-cased words in one pass

```
LongAdder adder = new LongAdder();

long distinctCount = reader.lines()
  .flatMap(line -> Stream.of(line.split(REGEXP)))
  .map(String::toLowerCase)
  .peek(s -> adder.increment())
  .distinct()
  .count();
```

Exercise 19

Compute the value of 21! (which needs to use BigInteger)

```
BigInteger result = IntStream.rangeClosed(1, 21)
  .mapToObj(n -> BigInteger.valueOf(n))
  .reduce(BigInteger.ONE, (m, n) -> m.multiply(n));


T merge(T identity, BiFunction accumulator)  ===

T result = identity;
for (T element : this stream)
  result = accumulator.apply(result, element)
return result;
```

ORACLE
Java™
CREATE
THE FUTURE