ORACLE

Lesson 2-5: Stream Interface: Intermediate Operations

Stream Interface

Overview

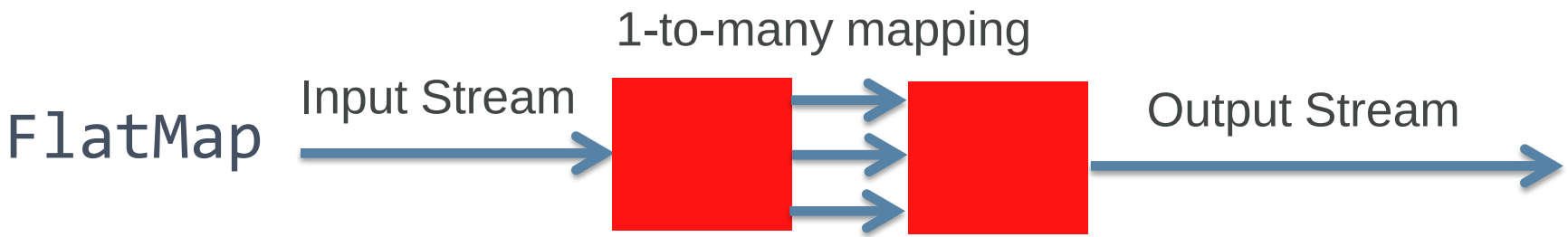
- A stream provides a sequence of elements
  - Supporting either sequential or parallel aggregate operations
- Most operations take a parameter that describes its behaviour
  - Typically using a Lambda expression
  - Must be non-interfering (does not modify the stream)
  - Typically stateless
- Streams may be changed from sequential to parallel (and vice-versa)
  - All processing is done either sequentially or in parallel
  - Last call wins

Filtering And Mapping

- `distinct()`
  - Returns a stream with no duplicate elements
- `filter(Predicate p)`
  - Returns a stream with only those elements that return true for the `Predicate`
- `map(Function f)`
  - Return a stream where the given `Function` is applied to each element on the input stream
- `mapToInt()`, `mapToDouble()`, `mapToLong()`
  - Like `map()`, but producing streams of primitives rather than objects

Maps and FlatMaps

Map Values in a Stream



FlatMap Example

Words in a File

```
List<String> output = reader
   .lines()
   .flatMap(line -> Stream.of(line.split(REGEXP)))
   .filter(word -> word.length() > 0)
   .collect(Collectors.toList());
```

Restricting The Size Of A Stream

- `skip(long n)`
  - Returns a stream that skips the first *n* elements of the input stream
- `limit(long n)`
  - Returns a stream that only contains the first *n* elements of the input stream

```
String output = bufferedReader
   .lines()
   .skip(2)
   .limit(2)
   .collect(Collectors.joining());
```

Sorting and Unsorting

- `sorted(Comparator c)`
  - Returns a stream that is sorted with the order determined by the `Comparator`
  - `sorted()` with no arguments sorts by natural order
- `unordered()`
  - Inherited from `BaseStream`
  - Returns a stream that is unordered (used internally)
  - Can improve efficiency of operations like `distinct()` and `groupingBy()`

Observing Stream Elements

As They Go Past

- `peek(Consumer c)`
  - Returns an output stream that is identical to the input stream
  - Each element is passed to the `accept()` method of the `Consumer`
  - The `Consumer` must not modify the elements of the stream
  - Useful for debugging and doing more than one thing with a stream

Section 5

Summary

- Stream interface represents aggregate operations on elements
- Most methods can use Lambda expressions to define behaviour
- Powerful range of intermediate operations allow streams to be manipulated as required
  - Build up complex processing from simple building blocks


ORACLE®