Lesson 3-6: Debugging Lambdas and Streams

Problems With Debugging Streams

- Streams provide a high level abstraction
  - This is good for making code clear and easy to understand
  - This is bad for debugging
    - A lot happens internally in the library code
    - Setting breakpoints is not simple
    - Stream operations are merged to improve efficiency

Simple Debugging

Finding What Is Happening Between Methods

- Use peek()
  - Like the use of print statements

```
List<String> sortedWords = reader.lines()              // Lines from file
   .flatMap(line -> Stream.of(line.split(REGEXP)))     // Words from file
   .map(String::toLowerCase)                           // In lower case
   .distinct()                                         // Remove duplicates
   .sort((x, y) -> x.length() - y.length())            // Sort by length
   .collect(Collectors.toList());                      // Collect to list
```

Simple Debugging

Finding What Is Happening Between Methods

- Use peek()
  – Like the use of print statements

```
List<String> sortedWords = reader.lines()                  // Lines from file
   .peek(System.out::println)                               // Print lines from file
   .flatMap(line -> Stream.of(line.split(REGEXP)))          // Words from file
   .map(String::toLowerCase)                                // In lower case
   .distinct()                                              // Remove duplicates
   .sort((x, y) -> x.length() - y.length())                 // Sort by length
   .collect(Collectors.toList());                           // Collect to list
```

Simple Debugging

Finding What Is Happening Between Methods

- Use `peek()`
  - Like the use of print statements

```
List<String> sortedWords = reader.lines()               // Lines from file
   .flatMap(line -> Stream.of(line.split(REGEXP)))      // Words from file
   .peek(System.out::println)                           // Print words
   .map(String::toLowerCase)                            // In lower case
   .distinct()                                          // Remove duplicates
   .sort((x, y) -> x.length() - y.length())             // Sort by length
   .collect(Collectors.toList());                       // Collect to list
```

Setting A Breakpoint

Using peek()

- Add a `peek()` method call between stream operations
- Use a `Consumer` that does nothing if required
  - Some debugging tools don't like empty bodies

```
List<String> sortedWords = reader.lines()
  .flatMap(line -> Stream.of(line.split(REGEXP))
  .peek(s -> s)
  .map(String::toLowerCase)
  .distinct()
  .sort((x, y) -> x.length() - y.length())
  .collect(Collectors.toList());
```

Set breakpoint here

No-op Lambda

Setting A Breakpoint

Using A Method Reference

- Lambda expressions do not compile to equivalent inner class
  - Compiled to invokedynamic call
  - Implementation decided at runtime
  - Better chance of optimisation, makes debugging harder
- Solution:
  - Extract the code from a Lambda expression into a separate method
  - Replace the Lambda with a method reference for the new method
  - Set breakpoints on the statements in the new method
  - Examine program state using debugger

Section 6

Summary

- Debugging is harder with Lambdas and streams
  - Stream methods get merged
  - Lambdas are converted to invokedynamic bytecodes and implementation is decided at runtime
  - Harder to set breakpoints
- `peek()` and method references can simplify things