Testing & Debugging Lambda Expressions

Raoul-Gabriel Urma
Richard Warburton

Testing & Debugging Overview

- Unit testing with Lambda Expressions
- Lambda Expressions as Test Doubles
- Debugging with laziness
- Breakpoints
- Stack Traces

Unit testing with Lambda Expressions

The Problem

- Lambda expressions don’t have a name or way of being externally referenced, so how do you test them?

- Two approaches
  - If if its very small don’t test in isolation
  - Use a method reference + a normal method

Small Example - Main Code

```
List<String> allToUpperCase(List<String> words) {
  return words.stream()
              // How do you test this lambda:
              .map(string -> string.toUpperCase())
              .collect(Collectors.toList());
}
```

Small Example - Test code

```
@Test
public void multipleWordsToUppercase() {
  List<String> input = Arrays.asList("a", "b", "hello");
  List<String> result = Testing.allToUpperCase(input);
  assertEquals(asList("A", "B", "HELLO"), result);
}
```

Larger Example - Main code

```
List<String> elementFirstToUpperCase(List<String> words) {
  return words.stream()
              .map(value -> {
                char firstChar =
                  Character.toUpperCase(value.charAt(0));
                return firstChar + value.substring(1);
              })
              .collect(toList());
}
```

Refactored Larger Example - Main code

```
List<String> elementFirstToUpperCase(List<String> words) {
  return words.stream()
              .map(this::firstToUppercase)
              .collect(toList());
}

String firstToUppercase(String value) {
  char firstChar = Character.toUpperCase(value.charAt(0));
  return firstChar + value.substring(1);
}
```

Refactored Larger Example - Test code

```
@Test
public void twoLetterStringConvertedToUppercase() {
  String input = "ab";
  String result = firstToUppercase(input);
  assertEquals("Ab", result);
}
```

The Solution

- If if its very small don’t test in isolation
- Use a method reference + a normal method

Lambda Expressions as Test Doubles

Mocking & Stubbing

- Lambda expressions can be test doubles for functional interfaces
  - Simpler than a framework for stubbing
  - Less so for mocking
- Whether you use it or not depend upon whether it makes sense in your codebase

Can be easier for stubbing

```
// Mockito:

AlbumProcessor processor = mock(AlbumProcessor.class);

when(processor.process(any())).thenReturn(x);

Object result = run(processor);

assertAboutResult(result);


// Lambdas:

Object result = run(arg -> x);

assertAboutResult(result);
```

Harder to verify interaction

```
AlbumProcessor processor = mock(AlbumProcessor.class);

read(processor);

verify(processor).process(notNull());


AtomicBoolean isCalled = new AtomicBoolean(false);
read(album -> {
 assertNotNull(album);
 isCalled.set(true);
});
assertTrue(isCalled.get());
```

Summary

- You can use lambda expressions for stubbing.

- They prove to be inelegant in case where you actually want to mock.

Debugging with laziness

Debugging with Laziness

- Streams are lazily evaluated
  - Things after in the code can be executed earlier at runtime!
  - Things earlier in the code can be executed after at runtime!
- Beware when you debug!

Let’s try it out debugging some streams!

Logging Example

```
Set<String> nationalities = new HashSet<>();
for (Artist artist : album.getMusicianList()) {
  if (artist.getName().startsWith("The")) {
    String nationality = artist.getNationality();
    System.out.println("Found nationality: " +
nationality);
    nationalities.add(nationality);
  }
}
return nationalities;
```

Logging Example with Streams

```
Set<String> nationalities
= album.getMusicians()
        .filter(artist -> artist.getName().startsWith("The"))
        .map(artist -> artist.getNationality())
        .peek(nation ->
          System.out.println("Found nationality: " + nation))
        .collect(toSet());
```

Breakpoints

Stack Traces

Stack Traces & Lambda Expressions

- Stack traces are a common debugging technique

- Gives you a complete information about:
  - what method was executed
  - What sequence of methods called the method
  - the error message in an exception

Example Code

```
public class StackTraceDebugging {

  public static void main(String[] args) {

    List<Point> points

      = Arrays.asList(new Point(12, 2), null);


    points.stream()

          .map(p -> p.getX())

          .forEach(System.out::println);

  }

}
```

Results in a NullPointerException

```
Exception in thread "main" java.lang.NullPointerException
at Debugging.lambda$main$0(Debugging.java:6)
at Debugging$$Lambda$5/284720968.apply(Unknown Source)
at java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline
.java:193)
at
java.util.Spliterators$ArraySpliterator.forEachRemaining(Spliterators
.java:948)
```

Method References (1)

```
// Example code:

points.stream().map(Point::getX).forEach(System.out::println);


// Console Output

at Debugging$$Lambda$5/284720968.apply(Unknown Source)

at java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline

.java:193)
```

Method References (2)

```
public static void main(String[] args) {
  List<Integer> numbers = Arrays.asList(1, 2, 3);
  numbers.stream()
         .map(Debugging::divideByZero)
         .forEach(System.out::println);
}
public static int divideByZero(int n){
  return n / 0;
}
```

```
at Debugging.divideByZero(Debugging.java:10)
at Debugging$$Lambda$1/999966131.apply(Unknown Source)
```

Summary

- Lambda Expressions can make stack traces less readable

- Method references in stack traces won't appear if it's a Null Pointer Exception on the object that is being referenced

Summary

Summary

- Unit test methods, not lambda expressions.
- You can use lambda expressions as test doubles
  - It’s only really a good idea for small stubs
- Lazy evaluation makes things harder to debug/log
  - Solvable with different tricks
- Stack Traces can be misleading

The End

Using mocks for answers

```
List<String> list = mock(List.class);
when(list.size()).thenAnswer(inv -> {
  kickOffProcess();
  return otherList.size();
});
assertEquals(3, list.size());
```