Get a Taste of Lambdas and Get Addicted to Streams

by Venkat Subramaniam

YouTube video link: <https://www.youtube.com/watch?v=1OpAgZvYXLQ>

[Rui Xue](https://www.youtube.com/channel/UCM_NRDl23u0PPf_qc3RS0LQ)[1 year ago (edited)](https://www.youtube.com/watch?v=1OpAgZvYXLQ&lc=UgzSyGkcZweK3aas0TV4AaABAg)

[7:00](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=420s) lambda [7:26](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=446s) backward compatibility of Java [8:18](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=498s) [9:34](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=574s) why Java 8 is backed by single abstract method interfaces (@FunctionalInterface) [9:47](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=587s) if an interface is single abstract method interface, you can use lambdas instead of anonymous inner classes [11:47](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=707s) how does it work under the hood [12:49](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=769s) illusion aobut lambda - syntax sugar, i.e. on the instruciton level, it is the same as anonymous inner class [14:16](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=856s) what lambda does under the hood [14:47](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=887s) what would the compiler be doing with lots of lambda (assumption only) \* [18:16](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=1096s) why it is not a good idea for lambdas to create extra classes like anonymous inner classes [19:14](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=1154s) a new feature since Java 7 - instruction - invokedynamic [20:53](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=1253s) in Java 8 struggling with the elimination of anonymous inner classes [21:00](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=1260s) the motivation: use invokedynamic to implement lambda expressions [25:16](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=1516s) example [29:18](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=1758s) internal iterator [39:00](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=2340s) method reference [44:28](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=2668s) avoid the urge of writing a large lambda expression - an anti-pattern [44:54](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=2694s) what are method references [45:41](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=2741s) examples of method references [47:15](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=2835s) static method reference [49:31](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=2971s) parameter as a target map(String::toString) [54:58](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=3298s) 2 arguments \* [56:23](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=3383s) the order of the arguments are important [57:38](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=3458s) another example - first parameter is the target, second parameter is the argument [59:33](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=3573s) limitations of method reference [1:08:00](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=4080s) filter [1:19:30](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=4770s) second part [1:29:07](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=5347s) stream is an abstraction [1:29:12](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=5352s) a stream is not a physical object with data [1:29:24](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=5364s) a stream is a bunch of functions you will evaluate eventually. There is no data sitting in a stream [1:29:48](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=5388s) a stream is a non mutating pipeline - stream is not mutating the data in a collection [1:30:31](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=5431s) stream functions [1:33:11](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=5591s) transforming function - [Stream.map](https://www.youtube.com/redirect?q=http%3A%2F%2Fstream.map%2F&stzid=UgzSyGkcZweK3aas0TV4AaABAg&event=comments&redir_token=kEYoLpvZmgIhu0S7xlBqS45QQlt8MTU1ODg5MjEyNUAxNTU4ODA1NzI1) [1:37:25](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=5845s) Stream.reduce() [1:43:47](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=6227s) InStream/DoubleStream.sum() - also a reduction operation [1:44:08](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=6248s) what is reduce operation - transform a collection into a concrete value or object [1:44:51](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=6291s) another reduce operation - collection [1:45:45](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=6345s) a wrong approach to do collect [1:47:00](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=6420s) explanation of the wrong code [1:47:18](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=6438s) shared mutability is devil [1:49:04](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=6544s) use Collectors.collect [1:51:17](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=6677s) Collectors.toMap [1:54:17](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=6857s) groupingBy [1:54:38](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=6878s) example [1:57:40](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=7060s) another example with mapping() method [1:59:37](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=7177s) performance [2:05:40](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=7540s) Q: How much work? [2:09:30](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=7770s) streams are LAZY [2:10:49](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=7849s) intermediate operations are postponed for evaluation [2:13:00](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=7980s) important \* [2:15:43](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=8143s) \* [2:16:15](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=8175s) [2:17:37](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=8257s) demo for performance analysis [2:19:36](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=8376s) lazy evaluations are only performed when the terminal operation is triggered [2:27:08](https://www.youtube.com/watch?v=1OpAgZvYXLQ&t=8828s) intermediate operation - Stream.sorted()

Rui Xue

1 year ago (edited)

7:00 lambda

7:26 backward compatibility of Java

8:18

9:34 why Java 8 is backed by single abstract method interfaces (@FunctionalInterface)

9:47 if an interface is single abstract method interface, you can use lambdas instead of anonymous inner classes

11:47 how does it work under the hood

12:49 illusion aobut lambda - syntax sugar, i.e. on the instruciton level, it is the same as anonymous inner class

14:16 what lambda does under the hood

14:47 what would the compiler be doing with lots of lambda (assumption only)

\* 18:16 why it is not a good idea for lambdas to create extra classes like anonymous inner classes

19:14 a new feature since Java 7 - instruction - invokedynamic

20:53 in Java 8 struggling with the elimination of anonymous inner classes

21:00 the motivation: use invokedynamic to implement lambda expressions

25:16 example

29:18 internal iterator

39:00 method reference

44:28 avoid the urge of writing a large lambda expression - an anti-pattern

44:54 what are method references

45:41 examples of method references

47:15 static method reference

49:31 parameter as a target map(String::toString)

54:58 2 arguments

\* 56:23 the order of the arguments are important

57:38 another example - first parameter is the target, second parameter is the argument

59:33 limitations of method reference

1:08:00 filter

1:19:30 second part

1:29:07 stream is an abstraction

1:29:12 a stream is not a physical object with data

1:29:24 a stream is a bunch of functions you will evaluate eventually. There is no data sitting in a stream

1:29:48 a stream is a non mutating pipeline - stream is not mutating the data in a collection

1:30:31 stream functions

1:33:11 transforming function - Stream.map

1:37:25 Stream.reduce()

1:43:47 InStream/DoubleStream.sum() - also a reduction operation

1:44:08 what is reduce operation - transform a collection into a concrete value or object

1:44:51 another reduce operation - collection

1:45:45 a wrong approach to do collect

1:47:00 explanation of the wrong code

1:47:18 shared mutability is devil

1:49:04 use Collectors.collect

1:51:17 Collectors.toMap

1:54:17 groupingBy

1:54:38 example

1:57:40 another example with mapping() method

1:59:37 performance

2:05:40 Q: How much work?

2:09:30 streams are LAZY

2:10:49 intermediate operations are postponed for evaluation

2:13:00 important

\* 2:15:43

\* 2:16:15

2:17:37 demo for performance analysis

2:19:36 lazy evaluations are only performed when the terminal operation is triggered

2:27:08 intermediate operation - Stream.sorted()