

Java-Success.com

Industrial strength Java/JEE Career Companion for those who want to go places

[Home](#)
[Java FAQs](#)
[600+ Java Q&As](#)
[Career](#)
[Tutorials](#)
[Member](#)
[Why?](#)
[Can u Debug?](#)
[Java 8 ready?](#)
[Top X](#)
[Productivity Tools](#)
[Judging Experience?](#)

[Home](#) › [Interview](#) › [Core Java Interview Q&A](#) › [Collection and Data structures](#) ›

02: ♦ Java 8 Streams, lambdas, intermediate vs terminal ops, and lazy loading with simple examples

02: ♦ Java 8 Streams, lambdas, intermediate vs terminal ops, and lazy loading with simple examples

Posted on [February 24, 2015](#) by [Arulkumaran Kumaraswamipillai](#)

A **stream** is an infinite sequence of consumable elements (i.e a data structure) for the consumption of an operation or iteration. Any **Collection<T>** can be exposed as a stream. It looks complex, but once you get it, it is very simple. The operations you perform on a stream can either be

1. Intermediate operations like map, filter, sorted, limit, skip, concat, substream, distinct, peek, etc producing another

[9 tips to earn more](#) | [What can u do to go places?](#) | **945+** members. [LinkedIn Group](#). [Reviews](#)

600+ Full Stack Java/JEE Interview Q&As ♥Free ♦FAQs

[open all](#) | [close all](#)

 [Ice Breaker Interview](#)

 [Core Java Interview C](#)

 [Java Overview \(4\)](#)

 [Data types \(6\)](#)

 [constructors-methc](#)

 [Reserved Key Wor](#)

 [Classes \(3\)](#)

 [Objects \(8\)](#)

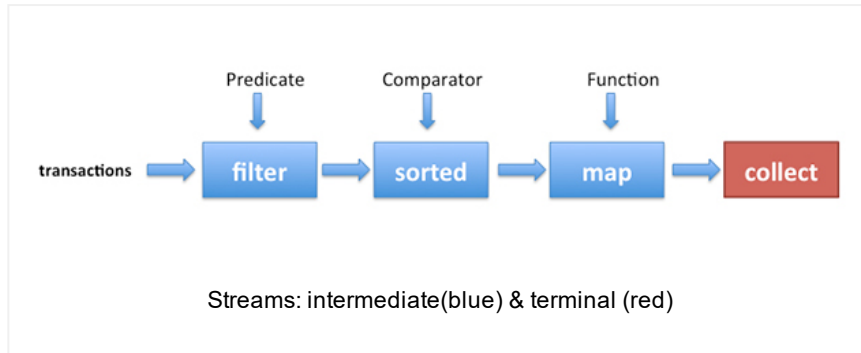
 [OOP \(10\)](#)

 [GC \(2\)](#)

 [Generics \(5\)](#)

java.util.stream.Stream<T> or a

2. Terminal operations like `forEach`, `reduce`, `collect`, `sum`, `max`, `count`, `matchAny`, `findFirst`, `findAny`, etc producing an object that is not a stream.



Basically, you are building a pipeline as in Unix. In Unix we “pipe” operations, and in Java 8, we stream them.

```
1 ls -l | grep "Dec" | Sort +4n | more
```

The **`stream()`** is a default method added to the **`Collection<T>`** interface in Java 8. The **`stream()`** returns a **`java.util.staream.Stream<T>`** interface with multiple abstract methods like `filter`, `map`, `sorted`, `collect`, etc. The **`DelegatingStream<T>`** is the implementing class.

Intermediate operations are **lazy operations**, which will be executed only after a terminal operation was executed. So when you call `.filter(i -> i % 3 == 0)` the lambda body isn't being executed at the moment. It will only be executed after a terminal operation was called (**`collect`**, in the example shown below). This is essential to understand from the viewpoint of adding break points in your IDE for debugging purpose.

Go through these examples to get a good handle on the stream concepts.

11 numbers 1 to 10 and an extra 6 are **a)** filtered first for multiples of 3 **b)** filtered for values less than 7 **c)** remove duplicates by adding to a `Set<T>` **d)** print the result.

FP (8)

01: ♦ 19 Java 8 I

02: ♦ Java 8 Stre

03: ♦ Functional

04: ♥♦ Top 6 tips

05: ♥ 7 Java FP

Fibonacci numbe

Java 8 String str

Java 8: What is c

IO (7)

Multithreading (12)

Algorithms (5)

Annotations (2)

Collection and Data

Differences Between

Event Driven Progr

Exceptions (2)

Java 7 (2)

Java 8 (24)

JVM (6)

Reactive Programn

Swing & AWT (2)

JEE Interview Q&A (3

Pressed for time? Jav

SQL, XML, UML, JSC

Hadoop & BigData Int

Java Architecture Inte

Scala Interview Q&As

Spring, Hibernate, & I

Testing & Profiling/Sa

Other Interview Q&A 1

Free Java Interview

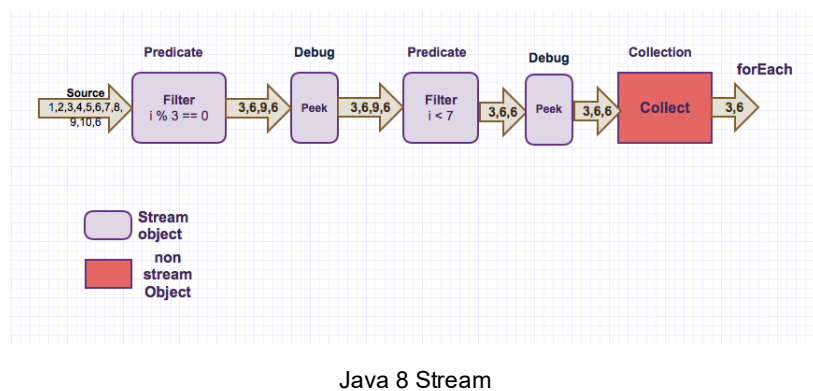
As a Java Architect

[Java architecture & design concepts](#)
[interview Q&As with diagrams](#) | [What should](#)

$i \rightarrow i \% 3 == 0$ is a **lambda expression** used as a predicate to filter only multiples of 3. So,

Q. what is this “lambda expression”?

A. In OOP or imperative programming, $x = x + 5$ makes sense, but in mathematics or **functional programming**, you can't say $x = x + 5$ because if x were to be 2, you can't say that $2 = 2 + 5$. In functional programming you need to say $f(x) \rightarrow x + 5$.



Example 1:

```

1 import java.util.Arrays;
2 import java.util.List;
3 import java.util.stream.Collectors;
4
5 public class Java8LambdaDebug {
6
7     public static void main(String[] args) {
8         List<Integer> list = Arrays.asList(1,2,3
9         list.stream()
10            .filter(i -> i % 3 == 0) //multiples o
11            .peek(i -> System.out.println("Debug p
12            .filter(i -> i < 7)
13            .peek(i -> System.out.println("Debug p
14            .collect(Collectors.toSet()) // remove
15            .forEach(i -> System.out.println("resu
16    }
17 }
18 }
19

```

In the above example, filter and peek are intermediate operations that return a “Stream<T>” object. The “peek” is used for **debugging**. The “collect(...)” is a terminal operation that returns a “Collection<T>” object, which extends

[be a typical Java EE architecture?](#)

Senior Java developers must have a good handle on

[open all](#) | [close all](#)

- [Best Practice \(6\)](#)
- [Coding \(26\)](#)
- [Concurrency \(6\)](#)
- [Design Concepts \(7\)](#)
- [Design Patterns \(11\)](#)
- [Exception Handling \(3\)](#)
- [Java Debugging \(21\)](#)
- [Judging Experience In](#)
- [Low Latency \(7\)](#)
- [Memory Management](#)
- [Performance \(13\)](#)
- [QoS \(8\)](#)
- [Scalability \(4\)](#)
- [SDLC \(6\)](#)
- [Security \(13\)](#)
- [Transaction Management](#)

80+ step by step Java Tutorials

[open all](#) | [close all](#)

- [Setting up Tutorial \(6\)](#)
- [Tutorial - Diagnosis \(2\)](#)
- [Akka Tutorial \(9\)](#)
- [Core Java Tutorials \(2\)](#)
- [Debugging Tutorial](#)
- [Java 8 Tutorial \(4\)](#)
- [02: ♦ Java 8 Streams](#)

“Iterable<T>” interface which has the “forEach(…)” method.
Don’t confuse this with the “forEach(…)” method in the
“java.util.stream.Stream<T>”.

Output:

```
1 Debug pt1: 3
2 Debug pt2: 3
3 Debug pt1: 6
4 Debug pt2: 6
5 Debug pt1: 9
6 Debug pt1: 6
7 Debug pt2: 6
8 result: 3
9 result: 6
10
```

Example 2:

Same as above, let’s introduce another terminal operation
sum().

```
1 import java.util.Arrays;
2 import java.util.List;
3
4 public class Java8LambdaDebug {
5
6     public static void main(String[] args) {
7         List<Integer> list = Arrays.asList(1,2,3
8
9         final int sum = list.stream()
10             .filter(i -> i % 3 == 0) //multiples o
11             .peek(i -> System.out.println("Debug p
12             .filter(i -> i < 7)
13             .peek(i -> System.out.println("Debug p
14             .mapToInt(Integer::intValue)
15             .sum(); //duplicate 6 is included 3+6+
16
17         System.out.println("sum=" + sum);
18     }
19 }
20
```

In the above example, filter, peek, and mapToInt are
intermediate operations that return a “Stream” object. “sum” is
terminal operation that returns a result.

Output:

[Learning to write](#)
[Top 6 Java 8 fea](#)
[Understanding J](#)
[JSON and Java Tu](#)
[XML and Java Tutc](#)
[CSV and Java Tutc](#)
[Hadoop & Spark Tuto](#)
[JEE Tutorials \(19\)](#)
[Scala Tutorials \(1\)](#)
[Spring & Hibernate Ti](#)
[Tools Tutorials \(19\)](#)
[Other Tutorials \(45\)](#)

Preparing for Java written & coding tests

[open all](#) | [close all](#)

[♦ Complete the given](#)
[Can you write code? \(](#)
[Converting from A to I](#)
[Designing your classe](#)
[Java Data Structures](#)
[Passing the unit tests](#)
[What is wrong with th](#)
[Writing Code Home A](#)
[Written Test Core Jav](#)
[Written Test JEE \(1\)](#)

How good are your...to go places?

[open all](#) | [close all](#)

[Career Making Know-](#)
[Job Hunting & Resum](#)

```
1 Debug pt1: 3
2 Debug pt2: 3
3 Debug pt1: 6
4 Debug pt2: 6
5 Debug pt1: 9
6 Debug pt1: 6
7 Debug pt2: 6
8 sum=15
9
```

Example 3:

Let's mix "intermediate" and "terminal" operations up.

```
1 import java.util.Arrays;
2 import java.util.List;
3 import java.util.stream.Collectors;
4
5 public class Java8LambdaDebug {
6
7     public static void main(String[] args) {
8         List<Integer> list = Arrays.asList(1,2,3
9
10         final int sum = list.stream()
11             .filter(i -> i % 3 == 0) //multiples o
12             .peek(i -> System.out.println("Debug p
13             .filter(i -> i < 7)
14             .peek(i -> System.out.println("Debug p
15             .collect(Collectors.toSet()) //remove
16             .stream()
17             .mapToInt(Integer::intValue)
18             .sum(); //duplicate is removed 3+6=12
19
20         System.out.println("sum=" + sum);
21     }
22 }
23
24
```

In the above example, `filter(..)`, `peek(..)`, `stream(..)`, and `mapToInt(..)` are intermediate operations that return a "Stream<T>" object. "collect(...)" and "sum()" are terminal operations. Since, "collect" returns a "Collection<T>" terminal object after removing the duplicate value of 6 with the help of `toSet()`, we need to call the `stream()` again to get the "Stream<T>" object back. Finally, "sum()" is a terminal operation.

Output:

```
1 Debug pt1: 3
2 Debug pt2: 3
```

```
3 Debug pt1: 6
4 Debug pt2: 6
5 Debug pt1: 9
6 Debug pt1: 6
7 Debug pt2: 6
8 sum=9
9
```

So, if still having trouble grasping this, have a look at the **Java 8 API docs** for Interfaces `Stream<T>`, `Iterable<T>` and Interface `Collection<E>`. Pay attention to default methods and return objects.

So, now with a little bit of help from the Java 8 API docs, you can perform different combination of operations on a collection of data. You can also debug by placing break points in your IDE like eclipse by keeping in mind that **intermediate ops are lazily evaluated** after a terminal operation. The **peek()** intermediate operation is very handy for debugging as well.

Popular Posts

♦ 11 Spring boot interview questions & answers

861 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

829 views

18 Java scenarios based interview Questions and Answers

448 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

407 views

♦ 7 Java debugging interview questions & answers

311 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

303 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

294 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

288 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

263 views

8 Git Source control system interview questions & answers

215 views

Bio

Latest Posts



Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via [Amazon.com](https://www.amazon.com) in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.



About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via [Amazon.com](https://www.amazon.com) in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

◀ ♦ Q29-Q53: Top 50+ JEE Interview Questions & Answers

Non-trivial Java 8 tutorial to appreciate the power of functional programming (FP) ▶

Posted in Collection and Data structures, FP, Java 8, Java 8 Tutorial

Tags: Core Java FAQs, Free Content

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ [Java generics in no time](#) ☀ [Top 6 tips to transforming your thinking from OOP to FP](#) ☀ [How does a HashMap internally work? What is a hashing function?](#)
☀ [10+ Java String class interview Q&As](#) ☀ [Java auto un/boxing benefits & caveats](#) ☀ [Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect](#)

Non-Technical Know Hows

☀ [6 Aspects that can motivate you to fast-track your career & go places](#) ☀ [Are you reinventing yourself as a Java developer?](#) ☀ [8 tips to safeguard your Java career against offshoring](#) ☀ [My top 5 career mistakes](#)

Prepare to succeed

☀ [Turn readers of your Java CV go from “Blah blah” to “Wow”?](#) ☀ [How to prepare for Java job interviews?](#) ☀ [16 Technical Key Areas](#) ☀ [How to choose from multiple Java job offers?](#)

© Disclaimer

The contents in this Java-Success are copy righted. The author has the right to correct or enhance the current content without any prior notice.

These are general advice only, and one needs to take his/her own circumstances into consideration. The author will not be held liable for any damages caused or alleged to be caused either directly or indirectly by these materials and resources. Any trademarked names or labels used in this blog remain the property of their respective trademark owners. No guarantees are made regarding the accuracy or usefulness of content, though I do make an effort to be accurate. Links to external sites do not imply endorsement of the linked-to sites.