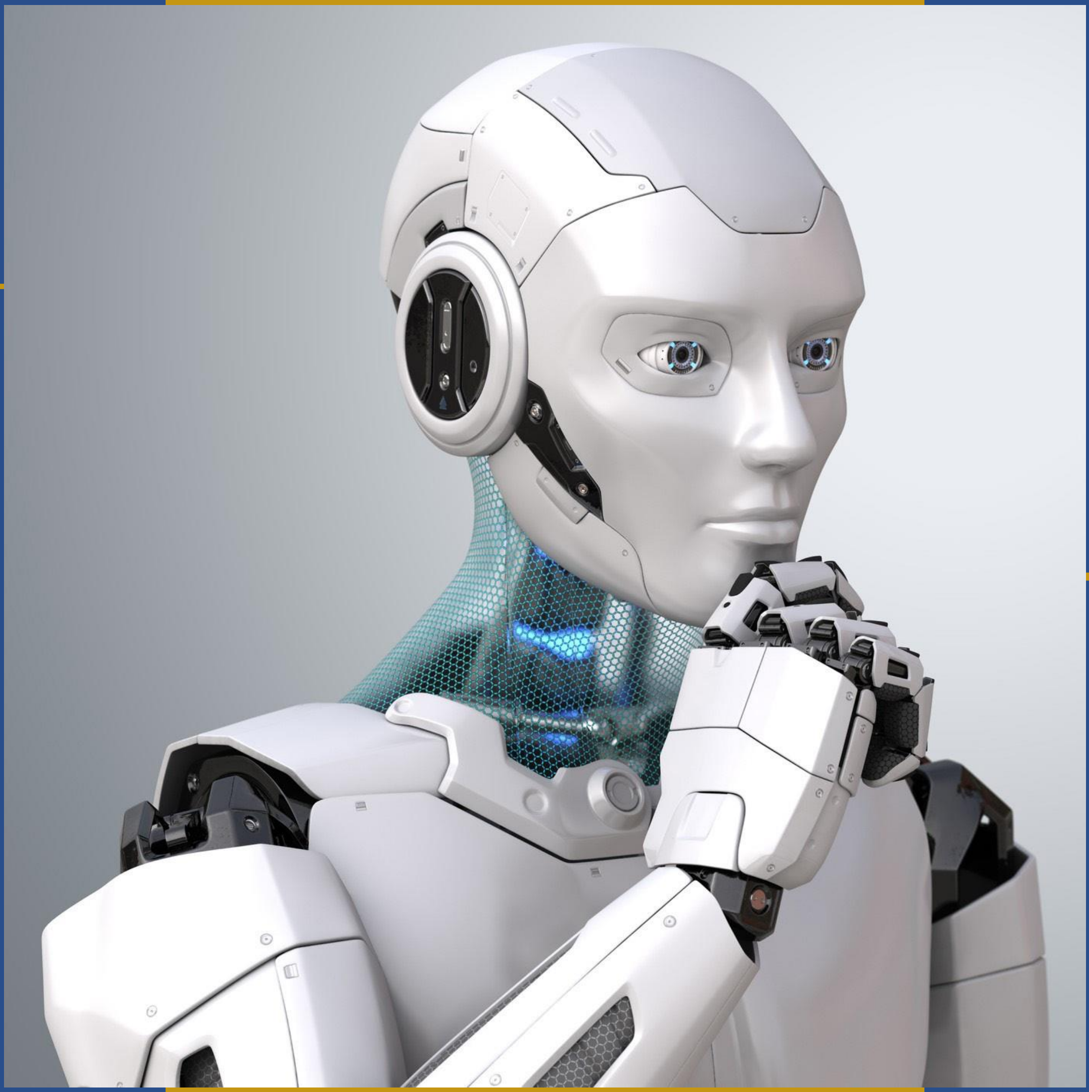
Java 12-Week Boot Camp

Week 1: Streams and Lambdas

PCWorkshops



Java Beginners: Streams and Lamdas
Including practical, illustrative coding examples

By: Sarah Barnard

2/3/2021

Java 12-Week Boot Camp

Streams and Lambdas

Course Notes and Exercises

Author: Sarah Barnard

Copyrights:© Sarah Barnard 2021

Acknowledgements: To Mary Smith who managed the formatting to Kindle format

Publishing House: PCW Courses Ltd, Pcworkshopslondon.co.uk

Content Summary

Streams and Lambdas

Java Language Keywords

- Here is a list of keywords in the Java programming language.
- You cannot use any of the following as identifiers in your programs.
- The keywords const and goto are reserved, even though they are not currently used.
- true, false, and null might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

abstract
assert***
boolean
break
byte
case
catch
char
class
const*
continue
default
do
double
else
enum****
extends
final
finally
float
for
goto*
if
implements
import
instanceof
int
interface
long
native
new
package
private
protected
public
return
short
static
strictfp**
super
switch
synchronized
this
throw
throws
transient
try
void
volatile
while

*	not used
**	added in 1.2
***	added in 1.4
****	added in 5.0

Convert List to Stream and Stream back to List

```
package week7_Lamdba;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;
import java.util.Set;
import java.util.TreeSet;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class a_01ConvertStreamToList {

/*  Convert Stream back to List
      We can accumulate the elements of the stream into a new List using a
Collector returned by Collectors.toList().
*/

// Program to convert stream to list in Java 8 and above
public static void main(String args[])
{
    List<String> cities = Arrays.asList("New York","New York",null,"Tokyo","New
Delhi");
    // convert a list to stream
       Stream<String> stream = cities.stream();

System.out.println("-- convert stream  ----");
    // convert Stream back to List, example 1
       cities = stream.collect(Collectors.toList());
       System.out.println(cities);


System.out.println("-- convert stream and  filter ----");
    // convert Stream back to List, example 2
       cities = stream.filter(Objects::nonNull).collect(Collectors.toList());
       System.out.println(cities);

System.out.println("--- convert to set to remove dups ---"  );
    // convert stream to set
    stream = cities.stream();  // redefinition of stream necessary
    Set<String> s1 = new TreeSet<String>();
    s1 = stream.collect(Collectors.toSet());
    System.out.println(s1);

    // https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-
summary.html
    }
}
```

Creating Streams

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;
import java.util.stream.Stream;

public class a_0CreateStreams {

public static void main(String[] args) {

// create a stream from fixed values
    Stream<String> stream4 = Stream.of("New York","Tokyo","New Delhi");

// create stream from an array
    String[] arr = new String[]{"New York","Tokyo","New Delhi"};
    Stream<String> stream3 = Arrays.stream(arr);

// create a stream from a list
    List<String> cities = Arrays.asList("New York","Tokyo","New Delhi");
    Stream<String> stream2 = cities.stream();

// create a stream from any collection
    BlockingQueue<Integer> q =  new ArrayBlockingQueue<Integer>(10);
    q.add(10);
    Stream<Integer> streamq = q.stream();

// create a stream from a list
     ArrayList<String> list = new ArrayList<>();
      list.add("Cat");
      list.add("Cheetah");
      list.add("");
      list.add("");
      Stream<String> stream1 = list.stream();
      // Paths.get(args) converts a string to a path

       }
}
```

Printing a Stream

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Random;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;
import java.util.stream.Stream;

public class a_1Print {

public static void main(String[] args) {

//          System.out.println("Arrays.toString -------------");
//          System.out.println(Arrays.toString(cities.toArray()));

//Print examples : printing a list

System.out.println("Printing a list -------------");
       List<String> cities = Arrays.asList("New York","Tokyo","New Delhi");
       for ( String val : cities) {
             System.out.println(val);
       }
    System.out.println(cities);

System.out.println("Printing a stream -------------");
//Stream<String> stream = listToStream(cities);
       Stream<String> stream = cities.stream();
       stream.forEach(System.out::println);


System.out.println("Printing a stream -------------");
       List<String> cities2 = Arrays.asList("London","Madrid","Paris");
       cities2.stream().forEach(System.out::println);

System.out.println("Printing a (blocking queue) stream -------------");
       BlockingQueue<Integer> q =  new ArrayBlockingQueue<Integer>(10);
       q.add(10);
       Stream<Integer> streamq = q.stream();
       streamq.forEach(System.out::println);

       }
}
```

Filter a Stream

```
import java.util.Arrays;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

public class a_3b_Filters {

	public static void main(String[] args) {
	List<String> strings=Arrays.asList("cat","fox", "", "cat", "wolf",
"abcd","", "jackal");
		for (String string : strings ) {
			System.out.println(string);
		}
		for (String string : strings ) {
			if (string.isEmpty()) {
				System.out.println(string);
			}
		}
		strings.stream().forEach(System.out::println);
// empties
	System.out.println("Print if  empty");
	strings.stream().filter(s->s.isEmpty()).forEach(System.out::println);
	System.out.println("Print if not empty");
	strings.stream().filter(s ->!s.isEmpty()).forEach(System.out::println);
//// String filters
	System.out.println("Print starts with a");
	strings.stream().filter(e->e.startsWith("a")).forEach(System.out::println);
	System.out.println("Print contains c");
	strings.stream().filter(e -> e.contains("c")).forEach(System.out::println);

//		//length of string
	System.out.println("Print length 3");
	strings.stream().filter(w -> w.length() == 3).forEach(System.out::println);

//// distinct values
	System.out.println("Print distinct");
	strings.stream().distinct().forEach(System.out::println);

////get list of unique numbers // distinct
	List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);
	numbers.stream().distinct().forEach(System.out::println);

//// filters
	System.out.println("all == 3" );
	 numbers.stream().filter( i -> i == 3 ).forEach(System.out::println) ;

	System.out.println("all bigger than 3" );
	numbers.stream().filter( i -> i > 3 ).forEach(System.out::println);

	System.out.println("count" );
	 System.out.println(numbers.stream().count());
	}
}
```

Filter Exercise

```
/*
```

Exercise 1

```
      read the file movies.csv
 * add each record(line) to a list
 * create a stream from the list and
 * if a value contains 'action', add it to the stream
 * print the stream
 *
*/
```

Exercise 2

```
Print a stream of all cats
      ArrayList<String> list = new ArrayList<>();
            list.add("cat");
            list.add("wild dog");
            list.add("wild cat");
```

Match in a Stream : Any , all, none

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class a_42_match {
// getting information from the whole stream
// is One in the List
// is Italy in the List
// is italy in all elements of the list
public static void main(String[] args) {
  ArrayList<String> list = new ArrayList<>();
  list.add("One");
  list.add("OneAndOnly");
  list.add("Derek");
  list.add("Change");
  list.add("factory");
  list.add("justBefore");
  list.add("Italy");
  list.add("Italy");
  list.add("Thursday");
  list.add("h");
  list.add("h");

  boolean isValid = list.stream().anyMatch(element -> element.contains("h"));
  boolean isValidOne = list.stream().allMatch(element -> element.contains("h"));
  boolean isValidTwo = list.stream().noneMatch(element -> element.contains("h"));
System.out.println("anyMatch h" +isValid + " allMatch h " + isValidOne + "
noneMatch h " + isValidTwo);

  boolean isValidz = list.stream().noneMatch(element -> element.contains("z"));
  System.out.println("z is nowhere noneMatch? " + isValidz);

  boolean isOne = list.stream().anyMatch(element -> element.contains("One"));
  boolean isItaly = list.stream().anyMatch(element -> element.contains("Italy"));
  boolean isAllItaly = list.stream().allMatch(e -> e.contains("Italy"));

  System.out.println("One is there anyMatch? " + isOne);
  System.out.println("Italy is there anyMatch? " + isItaly);
  System.out.println("Italy is everywhere allMatch? " + isAllItaly);

  isValidOne = list.stream().allMatch(element -> element.contains("h")); // false
  System.out.println("h is everywhere? " + isValidOne);
}
}
```

Exercise

```
      // is Harry in the list
      // is Harry in all elements of the list
      // is peter in the list
  String[] arr = new String[]{"Tom”,”Dick",”Harry”};
  Stream<String> stream3 = Arrays.stream(arr);
```

Reduce a Stream to one value

```
// REDUCE
/* Stream API allows reducing a sequence of elements to some value according
*  to a specified function.
*  with the help of the reduce() method of the type Stream.
             Imagine that you have a List<Integer> and you want to have a
             sum of all these elements and some initial Integer (in this example
23).
             So, you can run the following code and result will be 26 (23 + 1 + 1
+ 1).
*/

import java.util.Arrays;
import java.util.List;

public class a_43_ExampleReducs {

       public static void main(String[] args) {
              // reduce applies the operation on all values and the base value
              // to reduce the list to one value
           List<Integer> integers = Arrays.asList(2,2,2,3);
           // 2,2,2,3
           // 2*2 = 4*2 = 8 * 3 = 24 * 10
           Integer reduced = integers.stream().reduce(1, (a, b) -> a * b);
           System.out.println(reduced);

           // 2+2 = 4+2 = 6+3 = 9+10
           reduced = integers.stream().reduce(0, (a,b) -> a + b);
           System.out.println(reduced);
       }
}
```

Create a Stream of Random Numbers

```
//random.ints() returns datatype Instream , not Stream<Integer>

//https://en.wikipedia.org/wiki/Linear_congruential_generator

//ints()Returns an effectively unlimited stream
//of pseudorandom int values.
//The ‘limit’ method is used to reduce the size of the stream.
//The following code segment shows how to print 10 random
//numbers using limit.

import java.util.Random;
import java.util.stream.IntStream;

public class a_44Random {

public static void main(String[] args) {

   Random random = new Random();
   random.ints().limit(10).forEach(System.out::println);

   System.out.println("-----");
   // create a SORTED list of 10 random numbers
   Random r = new Random();
   r.ints().limit(10).sorted().forEach(System.out::println);

   System.out.println("-----");
              IntStream stream ;
                 stream = random.ints();
                 stream.limit(20).forEach(System.out::println);
 }

}
```

Count the Stream or a filtered Stream

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class a_45Counts {

public static void main(String[] args) {
//     filter and count
//     The ‘filter’ method is used to eliminate elements based on a criteria.
//  It then counts the remaining values.

   List<String> strings = Arrays.asList("cat", "", "bat", "vat", "rat","", "hat");

   long countall = strings.stream().count();
 //get count of empty strings
   long countNotEmpty = strings.stream().filter(s ->!s.isEmpty()).count();
   int  countEmpty = (int) strings.stream().filter(e -> e.isEmpty()).count();
   int  counta = (int) strings.stream().filter(e -> e.startsWith("a")).count();
   long countb = strings.stream().filter(e -> e.contains("b")).count();
   System.out.println("Notempty and Empty " +countNotEmpty + " " + countEmpty );
   System.out.println("All "+countall);
   System.out.println("Count start a "+counta + " start b" + countb);

   System.out.println("Contains a "+strings.stream().filter(e ->
e.contains("a")).count());


//length of string
 long count = (int) strings.stream().filter(w -> w.length() == 3).count();
 System.out.println("Strings of length 3: " + count);

}
}
```

Map each value in a Stream to get a different correlating Stream

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class a_46Map {

      public static void main(String[] args) {
//              map
//              The ‘map’ method is used to map each element to its corresponding
result. The following code segment prints unique squares of numbers using map.
   List<Integer> numbers = Arrays.asList(3, 2, 2, 3, 7, 3, 5);
   //get list of unique squares // distinct
   System.out.println("numbers " + numbers);
   numbers.stream().map( i -> i*2).forEach(System.out::println);

   List<Integer> newM = numbers.stream().map( i ->
i*+3/2).collect(Collectors.toList());
   System.out.println("Map: i + 3/2" +  newM);

   // distinct
   List<Integer> squaresList = numbers.stream().map( i ->
i*i).distinct().collect(Collectors.toList());
   System.out.println("squaresList " + squaresList);

   // Doubles
   List<Double> inputValues = Arrays.asList(3.0,2.0,2.0,3.0,7.0,3.0,5.0);
   List<Double> inputValues2 = inputValues.stream().map( i ->
i*i).distinct().collect(Collectors.toList());
   System.out.println(inputValues2);

   inputValues2 = inputValues.stream().map( i ->
i*3/2).collect(Collectors.toList());
   System.out.println(inputValues2);

// Stream<String> stream = listToStream(cities);
   System.out.println("------------");
       List<Integer> nrs = Arrays.asList(1,2,3,4,5,6,7,8,9,10);
       Stream<Integer> nstream = nrs.stream();
       nstream.map(x->x*x).forEach(y->System.out.println(y));

// String
 List<String> strings55 = Arrays.asList("cat", "", "bat", "vat", "rat","", "hat");
 strings55.stream().map(e -> e.toUpperCase()).forEach(System.out::println);
 strings55.stream().map(e -> e.replace("a","YY")).forEach(System.out::println);
       }

}
```

Map Exercise

```
// Celsius (°C) = (Fahrenheit - 32) / 1.8
// Use a stream and map the celcius values and print the fahrenheit values

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

        List<Double> fahrenheit = Arrays.asList(13.0, 12.0, 11.0, 13.0, 7.0, 1.0, 5.0);
```

Use the list of Celcius values and create a Stream of Fahrenheit values

Collectors

```
//Collectors
//Collectors are used to combine the result of processing on the elements
//of a stream.
//Collectors can be used to return a list or a string.
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class a_47Collectors {
public static void main(String[] args) {
// create a String
   List<String> stringsb = Arrays.asList("Peter", "Sally", "John");
   String s = stringsb.stream().collect(Collectors.joining(", "));
   System.out.println("Merged String: " + s);

// create Lists ( or Sets etc)
   List<String> strings4 = Arrays.asList("abc", "", "bc", "efg", "abcd","",
"jkl");
   // List to Stream
   Stream s1 = strings4.stream();

// Stream to List
// use .collect(Collectors.toList());
   List<String> mylist = strings4.stream().collect(Collectors.toList());
   // length of List / Stream
   System.out.println(strings4.size());

   // create a new list of filtered values
   List<String> stringsa = Arrays.asList("abc", "", "bc", "efg", "abcd","",
"jkl");
   List<String> filtered = stringsa.stream().filter(val ->
!val.isEmpty()).collect(Collectors.toList());
   System.out.println(filtered);

   // update the list stringsa with filtered values only
   stringsa= stringsa.stream().filter(ss ->
!ss.isEmpty()).collect(Collectors.toList());
   stringsa.forEach(System.out::println);

//create one String variable     from all values in a stream
// collect(Collectors.joining(","));
    List<String> letters = Arrays.asList("C","a","t");
       String word = letters.stream().collect(Collectors.joining(","));
       System.out.println(word);
        word = letters.stream().collect(Collectors.joining(" "));
       System.out.println(word);
        word = letters.stream().collect(Collectors.joining(""));
       System.out.println(word);

    String mergedString = stringsa.stream().collect(Collectors.joining(", "));
    System.out.println("Merged String: " + mergedString);

     mergedString = stringsa.stream().filter(string -
>!string.isEmpty()).collect(Collectors.joining(", "));
    System.out.println("Merged String: " + mergedString);
}  }
```

Sorting Stream

```
import java.util.Arrays;
import java.util.List;
import java.util.Random;

public class a_48Sort {

public static void main(String[] args) {
   //sort
   System.out.println("---  sort -");
   List<String> stringsA = Arrays.asList("abc", "", "bc", "efg", "abcd","",
"jkl");
   stringsA.stream().sorted().forEach(System.out::println);

   System.out.println("--- filter and sort -");
   stringsA.stream().filter(e ->
!e.isEmpty()).sorted().forEach(System.out::println);

//              sorted
//              The ‘sorted’ method is used to sort the stream. The following code
segment shows how to print 10 random numbers in a sorted order.
   Random random2 = new Random();
   random2.ints().limit(10).sorted().forEach(System.out::println);

}
}
```

Sort Exercise

```
Sort this list in ascending order
            ArrayList<String> list = new ArrayList<>();
            list.add("One");
            list.add("OneAndOnly");
            list.add("Derek");
            list.add("Change");
            list.add("factory");
            list.add("justBefore");
            list.add("Italy");
            list.add("Italy");
            list.add("Thursday");
            list.add("");
            list.add("");
```

Parallel Streams

```
//Parallel Processing
//parallelStream is the alternative of stream for parallel processing.
//Normally any java code has one stream of processing,
///where it is executed sequentially.
//Whereas by using parallel streams, we can divide the code
//into multiple streams
//that are executed in parallel on separate cores
//and the final result is the
//combination of the individual outcomes.
//The order of execution, however, is not under our control.
//Take a look at the following code segment that prints a
///count of empty strings using parallelStream.
```

```java
import java.util.Arrays;
import java.util.List;

public class a_51ParallelStream {

public static void main(String[] args) {

   List<String> strings3 = Arrays.asList("abc", "", "bc", "efg", "abcd","",
"jkl");
   //get count of empty string
   long count4 = strings3.parallelStream().filter(string3 ->
string3.isEmpty()).count();
   strings3.stream();
   System.out.println(count4);
   //It is very easy to switch between sequential and parallel streams.


// parallel processing ( instead of stream )
// breaks the stream into small subsets, run them in sequence in memory
// saving processing time
   int count = (int) strings3.parallelStream().filter(string ->
string.isEmpty()).count();
   System.out.println("Empty Strings: " + count);
}
}
```

Statistics: min, max,sum,average, count

//Statistics
//With Java 8, statistics collectors are introduced to calculate all
  //statistics when stream processing is being done.\

  // intStream are primitive int's and Stream not
  // Integer = int but can do more than int
  // int

```
import java.util.Arrays;
import java.util.IntSummaryStatistics;
import java.util.List;

public class a_5Stats {

public static void main(String args[])
{
    List<Integer> integers = Arrays.asList(1,2,13,4,15,6,17,8,19);
    IntSummaryStatistics testing = integers.stream().mapToInt((x) -
>x).summaryStatistics();

    System.out.println("List: " +integers);
    IntSummaryStatistics stats = integers.stream().mapToInt((x) -
>x).summaryStatistics();
//mapToInt Returns an IntStream consisting of the results of applying the given
function to the elements of this stream.
    System.out.println("Highest number in List : " + stats.getMax());
    System.out.println("Lowest number in List : " + stats.getMin());
    System.out.println("Sum of all numbers : " + stats.getSum());
    System.out.println("Average of all numbers : " + stats.getAverage());
    System.out.println("Count the values in the stream : " + stats.getCount());

}
}
```

Predicate

```
/*In Java 8, Predicate is a functional interface,
which accepts an argument and returns a boolean.
Usually, it used to apply in a filter for a collection of objects.
It makes it possible to build complexity into a filter using a Stream
*/
```

```
import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Stream;

public class a_7aPredicate {

// Program to convert a list to stream and filter it in Java 8 and above
public static void main(String args[]) {
        // working with a list
         System.out.println("Example 1-------------");
    List<String> cities = Arrays.asList("New York","Tokyo","New Delhi");
    for ( String s : cities ){
       if (s.startsWith("N")) {
             System.out.println(s);
       }
    }
    System.out.println("Example 2-------------");
    // working with a stream
    cities.stream().filter(c -> c.startsWith("N")).forEach(System.out::println);

//// with predicate
//// anonymous class
    System.out.println("Example 3-------------");
    Predicate<String> predicate = new Predicate<String>() {
        @Override
        public boolean test(String s) {
            // filter cities that start with "N"
            return s.startsWith("N"); // RETURNS A BOOLEAN
        }
    };
//// using predicate with  a list
    for ( String s : cities ){
       if (predicate.test(s)){
             System.out.println(s);
       }
    }
//// using predicate with a stream
    System.out.println("Example 4-------------");
    cities.stream().filter(predicate).forEach(System.out::println);
}
}
```