05: ♥ 7 Java FP (lambda expressions) real life examples in wrangling normal & big data

Posted on May 21, 2016 by Arulkumaran Kumaraswamipillai

This post extends Transforming your thinking from OOP to FP. In Big-data, functional programming is prevalent when working with data sets. For example, writing a Spark job to work with RDDs (Resilient Distributed Data sets).

In **Imperative** (E.g. OOP, procedural programming, etc) programming you can say

```
1 x = x + 5
```

```
2 y = x + y
3
```

where you are assigning x to x + 5. if x were to b 2, then after assignment it becomes 7 (i.e. 2 + 5)

In **functional programming** (FP), you can't say "x = x + 5" why? if x were to be 2, "2 = 2 + 5" is **wrong**. FP does not have assignment statements. FP is all about computation as the evaluation of "**mathematical functions**" and avoids changing-states and mutablity. In FP, you need to say

```
1 f(x) -> x + 5
2 f(x,y) -> x + y
3
```

where f(x) and f(x,y) are functions. Similarly, in the example below "(el1, el2) -> el1 + "," + el2″ is a **lambda expression** in FP. Where "el1" and "el2" are consecutive elements in a given list.

Example 1: Reducing a list of strings to CSV

stream -> reduce -> get

```
1  import java.util.Arrays;
2  import java.util.List;
3
4  public class Technology {
5
6      public static void main(String[] args) {
7          List<String> technologies = Arrays.asLis
8          // Java 8 FP
9          String csvTechnologies = technologies.st
10                .reduce((el1, el2) -> el1 + "," +
11                .get();
12         System.out.println(csvTechnologies);
13     }
14 }
15
```

Output

```
1
2 Java,JEE,JDBC,Spring,Hibernate
3
```

Example 2: Reducing a list of Integers to string (e.g. CSV)

stream -> map -> reduce -> get

```
1  import java.util.Arrays;
2  import java.util.List;
3
4  public class Weights {
5
6      public static void main(String[] args) {
7          List<Integer> weights= Arrays.asList(25,
8          // Java 8 FP
9          String csvWeights = weights.stream()
10                 .map(el1 -> el1.toString())
11                 .reduce((el1, el2) -> el1 + "," 
12                 .get();
13         System.out.println(csvWeights );
14     }
15 }
16
```

```
1
2  25,32,45,66,77
3
```

Example 3: Converting a List of unique objects to a map

Key=name -> **value**=Employee

```
1  import java.util.Arrays;
2  import java.util.List;
3  import java.util.Map;
4  import java.util.stream.Collectors;
5
6  public class ListToMap {
7
8      public static void main(String[] args) {
9          List<Employee> employees = Arrays.asList
10                 new Employee("Sam", 35, "English
11                 new Employee("Alice", 42, "Scien
12         // Java 8 FP
13         //Assume that names are unique, and use
14         Map<String, Employee> mapEmployees = emp
15                                             .collect(C
16                                             .toMap(emp
17         System.out.println(mapEmployees);
18     }
19
20     //inner pojo class
21     static class Employee{
22         private String name;
23         private int age;
24         private String department;
25
26         public Employee(String name, int age, St
```

be a typical Java EE architecture?

```
27              super();
28              this.name = name;
29              this.age = age;
30              this.department = department;
31          }
32
33          //getters & setters
34          public String getName() {
35              return name;
36          }
37
38          public int getAge() {
39              return age;
40          }
41
42          public String getDepartment() {
43              return department;
44          }
45
46          //toString
47          @Override
48          public String toString() {
49              return "Employee [name=" + name + ",
50          }
51      }
52  }
53
54
```

Output

```
1
2  {Alice=Employee [name=Alice, age=42, department=S
3
```

Example 4: Converting a List of non unique objects to a map

Key=name -> **value**=List<Employee>

```
1  import java.util.Arrays;
2  import java.util.List;
3  import java.util.Map;
4  import java.util.stream.Collectors;
5
6  public class ListToMap {
7
8      public static void main(String[] args) {
9          List<Employee> employees = Arrays.asList
10                 new Employee("John", 35, "Englis
11                 new Employee("Alice", 42, "Scien
12         // Java 8 FP
13         //Assume that names are NOT unique, and
14         //value will be a List of Employees
15         Map<String, List<Employee>> mapEmployees
16                                         .collect(C
17         System.out.println(mapEmployees);
18     }
19
```


Preparing for Java written & coding tests

How good are your...to go places?

```
20      //inner pojo class
21      static class Employee{
22          private String name;
23          private int age;
24          private String department;
25
26          public Employee(String name, int age, St
27              super();
28              this.name = name;
29              this.age = age;
30              this.department = department;
31          }
32
33          //getters & setters
34          public String getName() {
35              return name;
36          }
37
38          public int getAge() {
39              return age;
40          }
41
42          public String getDepartment() {
43              return department;
44          }
45
46          //toString
47          @Override
48          public String toString() {
49              return "Employee [name=" + name + ",
50          }
51      }
52  }
53
```

Output

```
1
2  {Alice=[Employee [name=Alice, age=42, department=
3
```

Example 5: Converting a Map keys to a List, sorted by values

entrySet -> stream -> sorted -> map -> collect

```
1   import java.util.Comparator;
2   import java.util.HashMap;
3   import java.util.List;
4   import java.util.Map;
5   import java.util.stream.Collectors;
6
7   public class MapToList {
8
9       public static void main(String[] args) {
10          Map<String, String> mapNameFaculty = new
11          mapNameFaculty.put("John", "Maths");
12          mapNameFaculty.put("Sam", "English");
13          mapNameFaculty.put("Alice", "Science");
```

```
14
15          //Convert to a List of names sorted by f
16          List<String> listOfNamesSortedByFaculty =
17                          .sorted(Comparator.compari
18                          .map(Map.Entry::getKey)
19                          .collect(Collectors.toList
20
21          System.out.println(listOfNamesSortedByFa
22      }
23  }
24
```

Output

```
1
2  [Sam, John, Alice]
3
```

Now, to sort by “length” of the faculty name

```
 1  import java.util.HashMap;
 2  import java.util.List;
 3  import java.util.Map;
 4  import java.util.stream.Collectors;
 5
 6  public class MapToList {
 7
 8      public static void main(String[] args) {
 9          Map<String, String> mapNameFaculty = new
10          mapNameFaculty.put("John", "Maths");
11          mapNameFaculty.put("Sam", "English");
12          mapNameFaculty.put("Alice", "Science");
13
14          //Convert to a List of names sorted by f
15          List<String> listOfNamesSortedByFaculty =
16                          .sorted((e1,e2) ->  e1.get
17                          .map(Map.Entry::getKey)
18                          .collect(Collectors.toList
19
20          System.out.println(listOfNamesSortedByFa
21      }
22  }
23
```

Output

```
1
2  [John, Alice, Sam]
3
```

FP is more memory intensive than imperative programming because in FP data is not overwritten, but sequences of versions are created to represent the data modification.

Nowadays, both the memory & disk are cheap. FP gives the programmer a lot more control about wrangling the data. Very useful in big data for functions like map, flatMap, reduce, combine, sort, etc. The "map" applies **a given function** to every data record on different machines in a cluster. This can be run in **parallel**. The "reduce" combines the individual results on different machines "by "applying a given function" to every data to reach a final result.

Example 6: Sum the list of numbers across the Hadoop cluster with Apache Spark

```
1
2 SparkConf conf = new SparkConf().setAppName("Sequ
3 JavaSparkContext sc = new JavaSparkContext(conf);
4
5 List<Integer> data = Arrays.asList(1, 2, 3, 4, 5)
6 JavaRDD<Integer> distData = sc.parallelize(data);
7 distData.reduce((a, b) -> a + b)
8
```

Example 7: Counting the number of blank lines in a given text input with Apache Spark.

More detailed explanation at: Apache Spark interview questions & answers

```
1
2  JavaRDD<String> lines = sc.textFile("data.txt");
3  final Accumulator<Integer> blankLines = sc.accum
4  JavaPairRDD<String, Integer> counts = lines.flat
5      {
6          if ("".equals(line)) {
7              blankLines.add(1); // increment the
8          }
9          return Arrays.asList(line.split(" "));
10     }).mapToPair(word -> new Tuple2<String, Integ
11       .reduceByKey((x, y) -> x + y);
12
13 System.out.println("Blank lines count: " + blank
14
```

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 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 in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

‹ 04: Convert XML file To an Avro File with Apache Spark – writing & reading

05: Convert XML file To an Avro File with avro-maven-plugin & Apache Spark ›

Posted in FP, Java 8

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?

Select Category

© Disclaimer

 ↑