Learning to write functional programming with Java 8 with examples

Posted on November 25, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Scenario 1: The ***Operation*** interface with the annotation *@FunctionalInterface*. This annotation ensures that you can only have a single abstract method. You can have additional default and static method implementations.

Step 1: Define the interface. It is an abstract method by default, and *@FunctionalInterface* ensures that you can only define a Single Abstract Method (aka SAM).

```
1 package com.java8.examples;
2
3 @FunctionalInterface
4 public interface Operation {
5     int operate(int operand1, int operand2);
6 }
7
```

Step 2: The ***OperationTest*** class that uses the functional interface with lambda expressions like (x, y) -> (x + y); and (x, y) -> (x * y) to add and multiply respectively. The method name *operate(int operand1, int operand2)* is not explictly called as the compiler knows to work it out from the parameters passed *int operand1*, *int operand2* which method to invoke.

```
1  package com.java8.examples;
2
3  public class OperationTest {
4
5   public static void main(String[] args) {
6     OperationTest test = new OperationTest();
7     System.out.println("add result = " + test.add
8     System.out.println("multiply result = " + tes
9   }
10
11  public int add( int input1,  int input2) {
12    Operation adder = (x, y) -> (x + y);
13    return adder.operate(input1, input2);
14  }
15
16  public int multiply( int input1,  int input2) {
17    Operation multiplier = (x, y) -> (x * y);
18    return multiplier.operate(input1, input2);
19  }
20 }
21
```

Output:

add result = 5
multiply result = 6

The above example will give the same output even without the *@FunctionalInterface* annotation. You could even have "**abstract**" in front of int operate(int operand1, int operand2);

```
1 package com.java8.examples;
```

```
2
3 public interface Operation {
4  abstract int operate(int operand1, int operand2)
5 }
6
7
```

Scenario 2: Have another *Operation2* interface that takes 3 integer parameters.

```
1 package com.java8.examples;
2
3 public interface Operation2 {
4  abstract int operate(int operand1, int operand2,
5 }
6
7
```

The *OperationTest* class that uses both interfaces. The *Operation* that take 2 integer parameters and *Operation2* that takes 3 integer parameters.

```
1  package com.java8.examples;
2
3  public class OperationTest {
4
5   public static void main(String[] args) {
6     OperationTest test = new OperationTest();
7     System.out.println("add result = " + test.add
8     System.out.println("multiply result = " + tes
9     System.out.println("add 3 numbers result = "
10  }
11
12  public int add( int input1,  int input2) {
13    Operation adder = (x, y)  -> (x + y);
14    return adder.operate(input1, input2);
15  }
16
17
18  public int multiply( int input1,  int input2) {
19    Operation subtracter =  (x, y) -> (x * y);
20    return subtracter.operate(input1, input2);
21  }
22
23  public int add2( int input1,  int input2, int i
24    Operation2 adder = (x, y, z)  -> (x + y + z);
25    return adder.operate(input1, input2, input3);
26  }
27
28 }
29
```

Output:

add result = 5
multiply result = 6
add 3 numbers result = 9

Scenario 3: Why do you have **default methods** in Java 8 interfaces?

This example demonstrates that default methods provide default behaviors, and closures are used to have special behaviors like add, multiply, divide, subtract, print, etc.

Step 1: Here is the revised *Operation* interface with default methods.

```
1  package com.java8.examples;
2
3  import java.util.function.BinaryOperator;
4  import java.util.function.Function;
5  import java.util.Objects;
6
7  @FunctionalInterface
8  public interface Operation<Intetger>  {
9
10     //SAM -- Single Abstract Method.
11     //identifier abstract is optional
12     Integer operate(Integer operand);
13
14     default Operation<Integer> add(Integer o){
15        return (o1) -> operate(o1) +   o;
16     }
17
18     default Operation<Integer> multiply(Integer
19       return (o1) -> operate(o1) * o;
20     }
21
22     //define other default methods for divide, s
23
24     default Integer getResult() {
25     return operate(0);
26     }
27
28   default void print(){
29    System.out.println("result is = " + getResult(
30   }
31
32 }
33
34
35
36
```

Step 2: The revised *OperationTest* class.

JEE Tutorials (19)
Scala Tutorials (1)
Spring & HIbernate Tu
Tools Tutorials (19)
Other Tutorials (45)

```
1  package com.java8.examples;
2
3  import static java.lang.System.out;
4
5  public class OperationTest {
6
7   public static void main(String[] args) {
8
9    Operation<Integer> calc = (x) -> (2);
10
11   Operation complexOp = calc.add(3)
12           .multiply(4)
13           .multiply(2)
14           .multiply(2)
15           .add(4);
16
17   complexOp.print();
18
19   int result = complexOp.getResult();
20  }
21
22 }
23
24
25
```

Output:

result is = 84

We know that Java does not support multiple implementation inheritance to solve the diamond problem (till Java 8). Java did only support multiple interface inheritance. That is, a class can implement multiple interfaces. By **having default method implementations in interfaces, you can now have** multiple behavioral inheritance in Java 8. Partially solving the diamond problem.

Scenario 4: Why do you have **static methods** in Java 8 interfaces?

The static methods are **helper methods**. Prior to Java 8, the Java APIs used to have separate interfaces and utility methods. For example, ***Collection*** interface and **Collections** utility class, ***Path*** interface and ***Paths*** utility class, etc.

So, instead of doing ***Collections.sort(list, ordering)***, in Java 8 APIs you could do ***list.sort(ordering)***;

Secondly, static helper methods are more expressive with meaningful names like ***plus5***, ***plus10***, etc as shown below.

```
1  package com.java8.examples;
2
3  import java.util.function.BinaryOperator;
4  import java.util.function.Function;
5  import java.util.Objects;
6
7  @FunctionalInterface
8  public interface Operation<Intetger>  {
9
10  //SAM -- Single Abstract Method.
11  //identifier abstract is optional
12  Integer operate(Integer operand);
13
14     default Operation<Integer> add(Integer o){
15        return (o1) -> operate(o1) +   o;
16     }
17
18     default Operation<Integer> multiply(Integer
19       return (o1) -> operate(o1) * o;
20     }
21
22     //define other default methods for divide, s
23
24     default Integer getResult() {
25     return operate(0);
26     }
27
28  default void print(){
29   System.out.println("result is = " + getResult(
30  }
31
32
33  //ads 5 to a given number
34  static Integer plus5(Integer input) {
35   return input + 5 ;
36  }
37 }
38
```

Now, how to invoke the static method via Lambda expressions:

```
1  package com.java8.examples;
2
3  import static java.lang.System.out;
4
5  public class OperationTest {
6
7   public static void main(String[] args) {
8
9    //plus5 is an expressive static helper method
```

```
10    Operation<Integer> calc = (x) -> Operation.plu
11
12    Operation complexOp = calc.add(3)
13            .multiply(4)
14            .multiply(2)
15            .multiply(2)
16            .add(4);
17
18    complexOp.print();
19
20    int result = complexOp.getResult();
21
22   }
23 }
24
25
```

Output:

result is = 164

Popular Posts

♦ 11 Spring boot interview questions & answers

825 views

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

766 views

18 Java scenarios based interview Questions and Answers

400 views

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

388 views

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

295 views

♦ 7 Java debugging interview questions & answers

293 views

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

285 views

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

279 views

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

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 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.**945+** paid members. join my LinkedIn Group. **Reviews**

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.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ 07: ♦ Java 8 API examples using lambda expressions and functional programming

Understanding Java 8 Streams and working with collections using Lambda expressions ›

Posted in Java 8, Java 8 Tutorial, member-paid

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

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

 ↑