Мульти-арные функции в Java

Напомню: арность (англ. arity) — это количество параметров функции. Соответсвенно мульти-арные (это слово можно писать вместе или раздельно) функции — это функции с несколькими параметрами. В Java 8 были введены функции с одним и двумя входными параметрами. А как быть если параметров больше?

Когда надо много входных параметров

В Java существует Function<X, R> и BiFunction<X, Y, R>, где X и Y это типы входных параметров, а R — тип выходного параметра. А вот функции с тремя и большим количеством входных параметров необходимо определять самому.

1. Можно заменить (*используя карринг - метод преобразования функции с N параметрами в функцию с N — 1 параметрами*) так BiFunction<X, Y, R> на Function<X, Function<Y, R>>. Такой финт позволяет реализовать любую арность только через интерфейс Function.

2. Или чтобы дальше определять функции с тремя входными параметрами:

r=f(a,b,c)

вам надо сначала однажды определить интерфейс:

@FunctionalInterface

public interface Function3Arity<A, B, C, R> {

R apply(A a, B b, C c);

}

После этого вы можете определять конкретные варианты тернарных (трех-арных) функций. Например вот так:

private static Function3Arity<Integer, String, Integer, String> f3 =

(a, op, b) ->{return "" + a + op + b + "=" + (a+b);};

Проверим, как это работает:

@Test

public void testFunction3Arity() {

String result = f3.apply(2, "+", 3);

assertEquals("2+3=5", result);

}

Соответствующие интерфейсы вы должны определить для каждой используемой арности N=3,4,…   
И все бы хорошо, да худо только в том, что прийти к этому решению рациональным путем просто невозможно. (Если Вы как и я не являетесь экспертом в области функционального программирования). Я имею ввиду способность метода apply воспринимать и правильно интерпретировать произвольное количество параметров. В документации это не написано. И не написано, можно ли это сделать как-нибудь другим способом. А я, когда передо мной возникла эта задача, надеялся найти нечто подобное в спецификации класса Function или содержащего её пакета.

Когда надо много выходных параметров

Мы рассмотрели пример, где было много входных параметров. А что делать, если у нас много выходных параметров?   
Как известно, Java позволяет с помощью return возвращать только один примитивный элемент или объект. А хотелось бы иметь возможность уже на уровне сигнатуры функции различать входные и выходные параметры. Т.е. иметь сигнатуры типа:

(c,d)=f(a,b)

К сожалению, сделать это напрямую не получится. Выходные параметры надо как-то структурировать. Для этого можно создавать временный объект, либо записывать параметры в список (List<?>). Первый способ тяжеловесен, а второй неприятен потерей статического контроля над типами выходных параметров, если эти типы разные.   
С моей точки зрения, более элегантным является использование <TupleN<X1,X2, ..Xn>.  
Например, класс Tuple2 выглядит вот так:

public class Tuple2<A, B> {

public final A a1;

public final B a2;

public Tuple2(A t, B u) {

a1 = Objects.requireNonNull(t);

a2 = Objects.requireNonNull(u);

}

@Override

public boolean equals(Object o) {…}

@Override

public int hashCode() {…}

}

С помощью этого класса функцию с тремя входными и двумя выходными параметрами можно определить вот так:

private static Function3Arity<Integer, String, Integer, Tuple2<Integer, String>> f3And2 =

(a, op, b) ->{

int intValue = a + b;

String sValue = "" + a + op + b + "=" + (a+b);

return new Tuple2<>(intValue, sValue);

};

Проверим как это работает:

@Test

public void testFunction3And2Arity() {

Tuple2<?,?> result = f3And2.apply(2, "+", 3);

assertEquals(5, result.a1);

assertEquals("2+3=5", result.a2);

}

Заключительное правило

1. Если в вашей функции 3 и больше входных параметра(ов) — вам необходимо определить новый N-арный интерфейс и с его помощью в последующем определять конкретные функции.
2. Если в вашей функции 2 и больше выходных параметра(ов) — определите класс TupleN и пакуйте в него параметры перед выводом из функции с помощью return.