Java Workbook

05. Stream API, Vol. II

~by Andrzej "Andret2344" Chmiel

Exercise 1: Powers of two

Create a lambda expression that for a given n returns 2^n. Use this lambda expression to create a stream of subsequent 11 subsequent powers of 2 (starting from $n = 0$) and print all them in separate lines, so the code prints out this:

```
1
2
4
8
16
32
64
128
256
512
1024
```

Exercise 2: Greatest common denominator

In the `main` method, create a lambda expression that calculates the GCD for two numbers. Use this lambda expression to find the GCD of the stream created in the `getIntStream()` method and to print it out.

```
class MainPopulation {
    public static void main(final String[] args) {

    }

    private static IntStream getIntStream() {
        return IntStream.of(12, 36, 96, 16, 24);
    }
}
```

For the given stream, the output should be:

```
4
```

Exercise 3: Constant and odd streams

Create a method that:

1. Accepts an `int` and returns an infinite stream of the given number.
2. Creates an infinite stream of odd numbers.

Exercise 4: Euclidean norm

Definition: On the n-dimensional Euclidean space $\mathbb{R}^2$, the intuitive notion of length of the vector $x = (x_1, x_2, x_3, ..., x_n)$ is captured by the formula $\|x\|_2 = \sqrt{x_1^2 + x_2^2 + x_3^2 + ... + x_n^2}$. The Euclidean norm is used in calculations of such things as the diameter of a square ($d = \sqrt{a^2 + b^2}$) or a cube ($d = \sqrt{a^2 + b^2 + c^2}$).

Generate a 10000-element stream of real numbers $r \in [0, 1]$ and treating it as a vector, count its Euclidean norm and print it.

Exercise 5: Fibonacci sequence

Definition: The Fibonacci sequence is a recursive sequence that starts with $x_1 = 0, x_2 = 1$ and has each element $x_n = x_{n-1} + x_{n-2}$.

Create a stream having elements from the Fibonacci sequence (starting from 0).

Exercise 6: List filtering

Given the following code:

```
record Product(String name, String type, double price) {
}

class MainProducts {
    public static void main(final String[] args) {
        final List<Product> productList = List.of(
                new Product("eggs", "food", 7.99),
                new Product("bike", "other", 1299.89),
                new Product("beer", "drink", 3.5),
                new Product("ham", "food", 4.85),
                new Product("apples", "food", 2.49),
                new Product("butter", "food", 6.99),
                new Product("cake", "food", 23.39));
        final double avg = countAverage(productList);
        System.out.printf("Average price of food: %.2f", avg);
    }

    private static double countAverage(final List<Product> productList) {
        return 0;
    }
}
```

Using the Stream API, fill in the `countAverage` method so that it counts an average price of the given products. So running the code prints this:

```
Average price of food: 9.14
```

Exercise 7: Weights

Given the following code:

```
record Item(String name, double weight) {
}

record Category(String name, List<Item> items) {
}

class Warehouse {
    private static final List<Category> categories = List.of(
            new Category("electronics", List.of(
                    new Item("tv", 6.11),
                    new Item("iron", 1.39))),
            new Category("parts", List.of(
                    new Item("motherboard", 0.56),
                    new Item("cpu", 0.13),
                    new Item("gpu", 0.67),
                    new Item("dvd_drive", 0.43),
                    new Item("hdd", 0.34))),
            new Category("accessories", List.of(
                    new Item("mouse", 0.32),
                    new Item("keyboard", 0.51),
                    new Item("webcam", 0.25))));

    public static void main(final String[] args) {
        System.out.println(search("electronics"));
        System.out.println(search("parts"));
        System.out.println(search("accessories"));
        System.out.println(search("toys"));
    }

    private static double search(final String name) {
        return -1;
    }
}
```

Using exactly one stream fill in the `search` method so that it finds and returns the heaviest item in a category of the given name or `-1` if the category is empty or invalid. So the code above prints this:

```
6.11
0.67
0.51
-1.0
```

Exercise 8: Stream's possibilities

Given the following code:

```
class MainStreamPossibilities {
    public static void main(final String[] args) {
        print2dArray(new String[][]{{"a", "b"}, {"c", "d"}, {"e", "f"}});
        System.out.println();
        print2dList(List.of(List.of(1, 2, 3, 4, 5), List.of(2, 3, 5, 7, 9)));
        System.out.println();
        printPrimitiveArray(new int[]{1, 2, 3, 4, 5, 6});
    }

    private static void print2dArray(final String[][] array) {

    }

    private static void print2dList(final List<List<Integer>> list) {

    }

    private static void printPrimitiveArray(final int[] array) {

    }
}
```

Fill in the `print2dArray`, `print2dList`, and `printPrimitiveArray` methods so each uses exactly one stream and only method references in it so that it prints out this:

```
abcdef
[1, 2, 3, 4, 5][2, 3, 5, 7, 9]
123456
```

Exercise 9: Prime numbers' stream

Generate a 10-element `IntStream` of unique random prime numbers $n \in [0, 2000]$ and print out them in descending order.

Exercise 10: Reflective method reference

Given the following code:

```
class MainFilteringStream {
    private interface Animal { }

    public static void main(final String[] args) {
        Stream.of("java", new Cat(), new Dog(), 1, "test", new Cat(), 2.0)
                .map(string -> String.format("I am: %s", string))
                .forEach(System.out::println);
    }

    private static class Cat implements Animal {
        @Override
        public String getName() {
            return "cat";
        }
    }

    private static class Dog implements Animal {
        @Override
        public String getName() {
            return "dog";
        }
    }
}
```

Not changing `Cat` and `Dog` classes, fix the code, so it compiles. Then add missing operations to the stream in the `main` method (do not change or remove existing ones) using only method references, so running the code prints out this:

```
I am: cat
I am: dog
I am: cat
```