CS2030 Lab #7 Java Stream Exercises

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

Java Streams

Topic Coverage

- Application of Java Streams

Requirements

- **Java Streams are to be used for the method implementations as specified in this assignment**

The Tasks

There are several tasks in this assignment.

In each task, you are to

- define a `Main` class with the method implementation(s) of the task(s)

Task 1

Twin Primes

A prime number is a natural number greater than 1 that is only divisible by 1 and itself. A **twin prime** is one of a pair of prime numbers with a difference of 2. For example, 41 and 43 are twin primes.

Define the method `twinPrimes` which takes in an integer n and returns an array of increasing twin primes from 0 until n inclusive.

```
static int[] twinPrimes(int n)
```

```
jshell> Main.twinPrimes(100)
$.. ==> int[15] { 3, 5, 7, 11, 13, 17, 19, 29, 31, 41, 43, 59, 61, 71, 73 }

jshell> Main.twinPrimes(2)
$.. ==> int[0] { }
```

Click here to submit to CodeCrunch.

Task 2

Greatest Common Divisor

Define the method `gcd` that takes in two positive integers `m` and `n` and returns the greatest common divisor using the Euclidean Algorithm.

```
static int gcd(int m, int n)
```

```
jshell> Main.gcd(539, 84)
$.. ==> 7
```

```
jshell> Main.gcd(84, 539)
$.. ==> 7

jshell> Main.gcd(1,1)
$.. ==> 1
```

Click here to submit to CodeCrunch.

Task 3

Counting Repeats

Define the method `countRepeats` that takes in an integer array of digits `0` to `9` and returns the number of occurrences of adjacent repeated digits. You may assume that there are at least three elements in the array.

```
static long countRepeats(int... array)
```

For example,

- the array `{0, 1, 2, 2, 1, 2, 2, 1, 3, 3, 1}` has three occurrences of repeated digits
- the array `{0, 1, 1, 1, 1, 2}` has one occurrence

The following is a sample run of the program. User input is underlined.

```
jshell> Main.countRepeats(0,1,2,2,1,2,2,1,3,3,1)
$.. ==> 3
```

Click here to submit to CodeCrunch.

Task 4

Normalized Mean

Given a list `T` of `n` integers t_i, the normalized value of each t_i is defined as

$$\bar{t_i} = \frac{t_i - \min_{\mathbf{T}}}{\max_{\mathbf{T}} - \min_{\mathbf{T}}}$$

where $\min_T$ and $\max_T$ represent the minimum and maximum values among all `n` values in `T`.

For example, the list of values `{1,2,3,4,5}` upon normalizing will become `{0,0.25,0.5,0.75,1}` since $\min_T = 1$ and $\max_T$=5.With the set of normalized values generated, the normalized mean can be easily computed to be 0.5.

Notice from the above that finding the normalized mean requires values in the list to be accessed twice: once for finding the maximum/minimum, and a second time to compute each normalized value and finding the mean.

Alternatively, we can re-expressed the normalized mean as

$$\bar{t}_{mean} = \frac{\sum_i \bar{t_i}}{n} = \frac{\sum_i \frac{t_i - \min_{\mathbf{T}}}{\max_{\mathbf{T}} - \min_{\mathbf{T}}}}{n} = \frac{\frac{\sum_i t_i - n\min_{\mathbf{T}}}{\max_{\mathbf{T}} - \min_{\mathbf{T}}}}{n} = \frac{\frac{\sum_i t_i}{n} - \min_{\mathbf{T}}}{\max_{\mathbf{T}} - \min_{\mathbf{T}}}$$

This way need to only access each element in the list exactly once.

Define the method `normalizedMean` that takes in a Stream of Integer elements and returns the normalized mean

```
static double normalizedMean(Stream<Integer> stream)
```

```
jshell> Main.normalizedMean(Stream.of(1, 2, 3, 4, 5))
$.. ==> 0.5

jshell> Main.normalizedMean(Stream.of(1, 1))
$.. ==> 0.0

jshell> Main.normalizedMean(Stream.of(1))
$.. ==> 0.0

jshell> Main.normalizedMean(Stream.of())
$.. ==> 0.0
```

Click here to submit to CodeCrunch.

Submission (Practice)

Your Files: BROWSE

SUBMIT (only .java, .c, .cpp, .h, .jsh, and .py extensions allowed)

To submit multiple files, click on the Browse button, then select one or more files. The selected file(s) will be added to the upload queue. You can repeat this step to add more files. Check that you have all the files needed for your submission. Then click on the Submit button to upload your submission.