Integer Streams

Topic Coverage

- Application of basic 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. You need to complete ALL the tasks. The tasks pre-supposes familiarity with the primitive type `IntStream` and hence some method signatures take primitive arrays as arguments. However, if you are familiar with object `Streams`, you are allowed to change the method parameter type from primitive array to `List` instead.

In each task, you are to

- define a `Main` class with the `main` method to handle input and output;
- check for output format correctness using the `diff` utility (see specific level for usage details). Note that only **one** test case is provided for this;
- save a copy of all source files into the appropriate level directory (see specific level for usage details).

Task 1

Perfect Numbers

Define the method `isPerfect` that takes in an integer `n` and determines if `n` is a perfect number.

A perfect number is a positive integer that is equal to the sum of its proper divisors. A proper divisor is a positive integer other than the number itself that divides the number evenly (i.e. no remainder).

For example, `6` is the smallest perfect number, because the sum of its proper divisors `1`, `2`, and `3` is equal to `6`. `8` is not a perfect number because `1 + 2 + 4` is not equal to `8`. Note that `1` is not a perfect number by definition.

```
static boolean isPerfect(int n)
```

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

```
6
```

```
true
8
false
```

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test1.in | diff - test1.out
```

Make a copy of your Java programs to the level directory by typing the Unix commands

```
$ jar cvf stream1.jar *.java
$ mkdir stream1
$ cp *.java stream1
$ cp stream1.jar stream1
```

Verify your jar archive using

```
$ jar tvf ~/stream1/stream1.jar
```

Task 2

Square Numbers

Define the method `isSquare` that takes in an integer `n` and determines if `n` is a square number.

A perfect square is the square of an integer. Examples are `9 (= 3×3)`, `4 (= 2×2)`, and `1 (= 1×1)`.

```
static boolean isSquare(int n)
```

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

```
81
true
80
false
```

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test2.in | diff - test2.out
```

Make a copy of your Java programs to the level directory by typing the Unix commands

```
$ jar cvf stream2.jar *.java
$ mkdir stream2
$ cp *.java stream2
$ cp stream2.jar stream2
```

Verify your jar archive using

```
$ jar tvf ~/stream2/stream2.jar
```

Task 3

Counting Repeats

Define the method `countRepeats` that takes in a `List` of digits `0` to `9` and returns the number of occurrences of adjacent repeated digits.

```
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.

```
0 1 2 2 1 2 2 1 3 3 1
^D
Number of occurrences: 3
```

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test3.in | diff - test3.out
```

Make a copy of your Java programs to the level directory by typing the Unix commands

```
$ jar cvf stream3.jar *.java
$ mkdir stream3
$ cp *.java stream3
$ cp stream3.jar stream3
```

Verify your jar archive using

```
$ jar tvf ~/stream3/stream3.jar
```

Task 4

Finding Variance

Define the method `variance` that takes in an integer array of elements and returns the variance of the elements. The variance of an array of x_i elements is defined as

$$\sigma^2 = \frac{\sum_{k=0}^{n-1}(x_k - \mu)^2}{n-1}$$

where μ is the mean (or average) of all n elements. Note that variance is only valid for an array of two or more elements.

Examples,

- `variance(IntStream.rangeClosed(1,6).toArray())` returns `OptionalDouble[3.5]`;
- `variance(IntStream.of().toArray())` returns `OptionalDouble.empty`
- `variance(IntStream.of(1).toArray())` returns `OptionalDouble.empty`

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

```
0 1 2 2 1 2 2 1 3 3 1
^D
Variance: OptionalDouble[0.8545454545454545]
```

```
^D
Variance: OptionalDouble.empty
```

```
1
^D
Variance: OptionalDouble.empty
```

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test4.in | diff - test4.out
```

Make a copy of your Java programs to the level directory by typing the Unix commands

```
$ jar cvf stream4.jar *.java
$ mkdir stream4
$ cp *.java stream4
$ cp stream4.jar stream4
```

Verify your jar archive using

```
$ jar tvf ~/stream4/stream4.jar
```

Task 5

Prime Factors

Write a method `factors` with that takes in an integer `x` and returns a `IntStream` consisting of the factors of `x`.

```
static IntStream factors(int x)
```

As an example, `factors(6)` should return the stream of integers `1, 2, 3, 6`.

Thereafter, write a method `primeFactors` that takes in an integer `x (> 1)` and returns a `IntStream` consisting of the prime factors of `x`.

A prime factor is a factor that is a prime number, excluding `1`. For instance, prime factors of `6` are `2` and `3`.

```
static IntStream primeFactors(int x)
```

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

```
6
Prime factors of 6 are: 2 3
```

```
12
Prime factors of 6 are: 2 3
```

```
731
Prime factors of 731 are: 17 43
```

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test5.in | diff - test5.out
```

Make a copy of your Java programs to the level directory by typing the Unix commands

```
$ jar cvf stream5.jar *.java
$ mkdir stream5
$ cp *.java stream5
$ cp stream5.jar stream5
```

Verify your jar archive using

```
$ jar tvf ~/stream5/stream5.jar
```