NUS | Computing
National University of Singapore

CodeCrunch

CS2030 (2410) Lab #1

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

Java Streams

Java streams provide us with a way to devise solutions to computation problems using a declarative pipelined approach that let's us focus on the goals (or sub-goals) of a task, rather than how to do the task.

The Tasks

There are several tasks in this assignment. For each task, you are to define the appropriate method(s) within `Main.java`. We do this so that you can have your code compiled before running in `jshell`.

The following skeleton `java` is given.

```
import java.util.stream.IntStream;
import java.util.stream.Stream;
import java.util.List;

void main() {}
```

Save your solutions and compile using

```
$ javac --release 21 --enable-preview java
Note: java uses preview feature of Java SE 21.
Note: Recompile with -Xlint:preview for details.
```

Once compiled successfully, you may test your solution using `jshell`, e.g.

```
$ jshell
|  Welcome to JShell -- Version 21.0.4
|  For an introduction type: /help intro

jshell> /open Main.java

jshell> twinPrimes(3).count()
$.. ==> 1
```

You may also write your tests within the `main` method in `Main.java`, e.g.

```
void main() {
    System.out.println(twinPrimes(3).count());
}
```

then compile and run your program.

```
$ javac --release 21 --enable-preview Main.java
Note: java uses preview feature of Java SE 21.
Note: Recompile with -Xlint:preview for details.

$ java --enable-preview Main
1
```

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 `IntStream` comprising of distinct twin primes from `2` to `n`.

```
IntStream twinPrimes(int n)
```

```
jshell> twinPrimes(100)
$.. ==> java.util.stream.IntPipeline$...

jshell> twinPrimes(100).boxed().toList()
$.. ==> [3, 5, 7, 11, 13, 17, 19, 29, 31, 41, 43, 59, 61, 71, 73]

jshell> twinPrimes(100).count()
15

jshell> twinPrimes(2).forEach(x -> System.out.println(x))

jshell> twinPrimes(2).count()
0

jshell> twinPrimes(3).forEach(x -> System.out.println(x))
3
```

In the last example, `3` is still listed as a twin prime although `5` is out of the range.

Task 2: Reverse String

Complete the method `reverse` that takes in a `String str` and returns the reverse of `str`.

```
String reverse(String str)
```

Hint:

- the individual characters of a string can be obtained using the `substring` method;
- two strings can be concatenated using the `+` operator;

```
jshell> String str = "abc"
str ==> "abc"

jshell> str.length()
str ==> 3

jshell> str.substring(2, str.length())
$.. ==> "c"

jshell> str.substring(2, str.length()) + "z"
$.. ==> "cz"
```

Note that `substring(i, j)` returns the sub-string from index `i` (inclusive) to just before index `j`. You should start by streaming the appropriate indices.

```
jshell> /open Main.java

jshell> reverse("orange")
$.. ==> "egnaro"

jshell> reverse("one two three")
$.. ==> "eerht owt eno"

jshell> reverse("")
```

```
$.. ==> ""

jshell> reverse("the quick brown fox jumps over the lazy dog.")
$.. ==> ".god yzal eht revo spmuj xof nworb kciuq eht"
```

Task 3: Counting Repeats

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

```
int countRepeats(List<Integer> list)
```

For example,

- the list `[0, 1, 1, 1, 1, 2]` has one occurrence
- the list `[0, 1, 2, 2, 1, 2, 2, 1, 3, 3, 1]` has three occurrences of repeated digits

Hint: You need only look at every three consecutive digits to decide if a repeat needs to be counted.

```
jshell> /open Main.java

jshell> countRepeats(List.of(0, 1, 1, 1, 1, 2))
$.. ==> 1

jshell> countRepeats(List.of(0, 1, 2, 2, 1, 2, 2, 1, 3, 3, 1))
$.. ==> 3

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