CS2030 Programming Methodology
Semester 1 2024/2025

Week of 26 – 30 August 2024
Problem Set #1 Suggested Guidance
Declarative Programming with Java Streams

1. The i^{th} omega number is the number of distinct prime factors for the number i (> 0). The first 10 omega numbers are $0, 1, 1, 1, 1, 2, 1, 1, 1, 2$.

   Write a method `omega` that takes in an integer `n` (> 0) and returns an `IntStream` containing the first `n` omega numbers.

   ```
   IntStream omega(int n)

   jshell> omega(10).forEach(x -> System.out.print(x + " "))
   0 1 1 1 1 2 1 1 1 2
   ```

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

   boolean isPrime(int n) {
       return n > 1 && IntStream
           .range(2, n)
           .noneMatch(x -> n % x == 0);
   }

   IntStream factors(int x) {
       return IntStream
           .rangeClosed(1, x)
           .filter(d -> x % d == 0);
   }

   IntStream primeFactors(int x) {
       return factors(x)
           .filter(d -> isPrime(d));
   }

   int countPrimeFactors(int x) {
       return primeFactors(x)
           .reduce(0, (a, b) -> a + 1);
   }

   IntStream omega(int n) {
       return IntStream.rangeClosed(1, n)
           .map(i -> countPrimeFactors(i));
   }

   void main() {
       omega(10).forEach(x -> System.out.print(x + " "));
   }
   ```

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

$ java --enable-preview omega
0 1 1 1 1 2 1 1 1 2
```

2. Write a method `dot` that takes in two integer arguments a and b with $a \leq b$, and returns the cartesian dot-product defined as follows:

$$\{i \cdot j \mid i \in S,\ j \in S\} \quad \text{where} \quad S = [a, b]$$

For example, if $a = 1$ and $b = 3$, then $S \in [1, 3]$ and the result is

$$\{1 \cdot 1,\ 1 \cdot 2,\ 1 \cdot 3,\ 2 \cdot 1,\ 2 \cdot 2,\ 2 \cdot 3,\ 3 \cdot 1,\ 3 \cdot 2,\ 3 \cdot 3\}$$

```
jshell> IntStream dot(int a, int b) {
   ...>     return IntStream.rangeClosed(a, b)
   ...>         .flatMap(i -> IntStream.rangeClosed(a, b)
   ...>             .map(j -> i * j));
   ...> }
|  replaced method dot(int,int)
```

```
jshell> dot(1,3).forEach(x -> System.out.print(x + " "))
1 2 3 2 4 6 3 6 9
```

Now write a method `product` that takes in two integer arguments a and b with $a \leq b$, and returns the paired cartesian product defined as follows:

$$\{(i, j) \mid i \in S,\ j \in S\} \quad \text{where} \quad S = [a, b]$$

Use the `Pair` record defined as follows:

```
jshell> record Pair<T,U>(T t, U u) {}
|  created record Pair

jshell> new Pair<Integer,Integer>(1, 3)
$.. ==> Pair[t=1, u=3]

jshell> product(1,3).toList()
$.. ==> [Pair[t=1, u=1], Pair[t=1, u=2], Pair[t=1, u=3],
Pair[t=2, u=1], Pair[t=2, u=2], Pair[t=2, u=3],
Pair[t=3, u=1], Pair[t=3, u=2], Pair[t=3, u=3]]
```

```
jshell> Stream<Pair<Integer,Integer>> product(int a, int b) {
   ...>     return IntStream.rangeClosed(a, b)
   ...>         .boxed()
   ...>         .flatMap(i -> IntStream.rangeClosed(a, b)
   ...>             .mapToObj(j -> new Pair<Integer, Integer>(i, j)));
   ...> }
|  created method product(int,int)
```

3. Write a method that returns the first n Fibonacci numbers as a `Stream<Integer>`.

   For instance, the first 10 Fibonacci numbers are $1, 1, 2, 3, 5, 8, 13, 21, 34, 55$.

   Hint: Use `Pair` to keep two items in the stream.

```
jshell> Stream<Integer> fibonacci(int n) {
   ...>    return Stream.iterate(new Pair<Intger,Integer>(1, 1),
   ...>          pr -> new Pair<Integer,Integer>(pr.u(), pr.t() + pr.u()))
   ...>       .map(pr -> pr.t())
   ...>       .limit(n);
   ...> }
|  created method fibonacci(int)

jshell> fibonacci(10).toList()
$.. ==> [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

   To handle bigger values and avoid overflow, you may consider using the `BigInteger` *class.*