CS2030 Programming Methodology
Semester 1 2019/2020

11 October 2019
Problem Set #6
Lambda and Streams

1. Write a method `omega` with signature `IntStream omega(int n)` that takes in an `int n` and returns a `IntStream` containing the first n omega numbers.

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

   The `isPrime` method is given below:

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

2. 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: Write an additional `Pair` class that keeps two items around in the stream

3. Write a method product that takes in two `List` objects `list1` and `list2`, and produce a `Stream` containing elements combining each element from `list1` with every element from `list2` using a `BiFunction`. This operation is similar to a Cartesian product.

   ```
   public static <T,U,R> Stream<R> product(List<? extends T> list1,
           List<? extends U> list2,
           BiFunction<? super T, ? super U, R> func)
   ```

   For example, the following program fragment

   ```
   List<Integer> list1 = new ArrayList<>();
   List<String> list2 = new ArrayList<>();

   Collections.addAll(list1, 1, 2, 3, 4);
   Collections.addAll(list2, "A", "B");

   product(list1, list2, (str1, str2) -> str1 + str2)
       .reduce("", (x, y) -> x + y + " ")
   ```

   gives the output

   ```
   1A 1B 2A 2B 3A 3B 4A 4B
   ```

4. You are given two functions $f(x) = 2 \times x$ and $g(x) = 2 + x$.

   (a) By creating an abstract class `Func` with a public abstract method `apply`, evaluate $f(10)$ and $g(10)$.

   (b) The composition of two functions is given by $f \circ g(x) = f(g(x))$. As an example, $f \circ g(10) = f(2+10) = (2+10) * 2 = 24$. Extend the abstract class in question 4a so as to support composition, i.e. `f.compose(g).apply(10)` will give 24.

   (c) Now re-implement question 4b as a functional interface `Func<T,R>`

5. **Currying** is the technique of translating the evaluation of a function that takes multiple arguments into evaluating a sequence of functions, each with a single argument, $g(x, y) = h(x)(y)$. Using the context of lambdas in Java, the lambda expression `(x, y) -> x + y` can be translated to `x -> y -> x + y`.

   Show how the use of appropriate functional interfaces can achieve the curried function evaluation of two arguments.

   Hint: If the lambda above looks intriguing, try replacing the lambda with anonymous inner classes instead to make sense of the scope of the variables `x` *and* `y`.