The learning outcome of this practical session is to practice Java Streams with advanced collectors. The objective consists in partitioning numbers into prime and nonprime.

1 Partitioning with a Standard Collector

Using a stream, write a *predicate* that tests whether a given candidate number is prime or not (from now on, external iterations with loops will be avoided).

1. The primality test consists in checking whether the candidate number is a multiple of any number between 1 and itself.

2. Write a method taking as argument an integer \( n \) and partitioning the first \( n \) natural numbers into prime and nonprime in an object of type \( \text{Map}(<\text{Boolean}, \text{List}<\text{Integer}>) \).

2 Partitioning with a Custom Collector

This first solution has one drawback: for each candidate, we check whether it is a multiple of any of the numbers between 2 and some upper bound, whereas we are interesting in checking only if it is a multiple of any of the prime numbers found so far between 2 and some upper bound. We now want to generate a \( \text{List}<\text{Integer}> \) of all primes.

1. Rewrite the function testing the primality by considering two arguments: a list of prime numbers found so far and the current candidate number to be tested.

2. Let’s define a collector that will use this function to process the prime numbers. In public interface \( \text{Collector}<T, A, R> \). What are \( T \), \( A \) and \( R \) in our case? Refer to the official documentation of this interface (select the one specific to the version of Java that is being used).

3. What are the supplier and the accumulator?

4. What are the last three functions?

5. What is the final code to generate all primes from 2 to \( n \)?

3 Optimization

Propose an optimization to speed up both predicates used above by improving the algorithm. To this end, remark that it is wasteful to test whether each number between 1 and the candidate is a divisor.