Real FizzBuzz

Please use Java for this assessment, with Maven as the build tool for all three projects.

Step 1:
Write some code that prints out the following for any contiguous range of numbers passed into the program:

* the number
* 'fizz' for numbers that are multiples of 3
* 'buzz' for numbers that are multiples of 5
* 'fizzbuzz' for numbers that are multiples of 15

e.g. if I run the program over a range from 1-20 I should get the following output
1 2 fizz 4 buzz fizz 7 8 fizz buzz 11 fizz 13 14 fizzbuzz 16
17 fizz 19 buzz
Zip source "FizzBuzz1" such that when extracted it can be imported as a Maven project then continue on to step two.

Step 2:
Enhance your existing FizzBuzz solution to perform the following:
* If the number contains a three you must output the text 'lucky'. This overrides any existing behaviour
e.g. if I run the program over a range from 1-20 I should get the following output
1 2 lucky 4 buzz fizz 7 8 fizz buzz 11 fizz lucky 14 fizzbuzz
16 17 fizz 19 buzz
Zip source as a separate file "FizzBuzz2" such that when extracted it can be imported as a Maven project then continue on to step three

Step 3:
Enhance your existing solution to perform the following:

* Produce a report at the end of the program showing how many times the following were output
** fizz
** buzz
** fizzbuzz
** lucky
** an integer

e.g. if I run the program over a range from 1-20 I should get the following output
1 2 lucky 4 buzz fizz 7 8 fizz buzz 11 fizz lucky 14 fizzbuzz

```
16 17 fizz 19 buzz
fizz: 4
buzz: 3
fizzbuzz: 1
lucky: 2
integer: 10
```
(Integer is 10 because there were 10 numbers that were not altered in any way).

Zip source as a separate file “FizzBuzz3” such that when extracted it can be imported as a Maven project.

Please include a readme on how to compile and run your code.

Send all three files to your TA contact or recruiter.

Guidelines for offline code tests:
You should not find this test to be particularly difficult. It is designed to be a straightforward coding exercise, and it should take you no more than a couple hours.

Things we are looking for:

- Test Coverage: The solution should be developed “test-first”, and should have excellent unit test coverage.
- Simplicity: We value simplicity as an architectural virtue and a development practice. Solutions should reflect the difficulty of the assigned task, and should not be overly complex. Layers of abstraction, patterns, or architectural features that aren’t called for should not be included.
- Self-explanatory code: The solution you produce must speak for itself. Multiple paragraphs explaining the solution are a sign that it isn’t straightforward enough to understand purely by reading code, and are not appropriate.
- Use of appropriate Java coding and project conventions.