CSC120 2025S Lab No. 7
Factoring

Instructor: Mitsu Ogihara

1 The Lab Objective

A positive integer `d` is a divisor of another positive integer `n` is `n` is a multiple of `d`; i.e., `n % d` equals `0`. A divisor `d` of `n` is non-trivial if the `d` is neither `1` nor `n`. A prime number is an integer without a non-trivial divisor.

This lab aims to write a Java class, `Factor`, that computes three things from each input integer it receives. The program receives, from the user, an indefinite number of input integers. If the input integer is < 2, the program terminates. Otherwise, the program executes the following:

- The program breaks the input number into a non-decreasing sequence of prime numbers whose product equals the input number.

  The numbers in the list are separated by a `String " * "`.

- The program breaks the input number into a non-decreasing sequence of prime numbers whose product equals the input number. Unlike the previous one, each repeated prime number is compressed into the prime number "raised to the power of" the number of appearances. The format is `pê` where `p` is the prime number and `e` is the exponent.

- The program finds every nontrivial divisor of the input number and produces a list of the numbers in increasing order, where `", "` separates between two divisors.

Next appears an execution example:

```
1
2  Enter an integer (an integer <= 1 will stop the program)
3  Your number: 363
4  The factorization is: 3 * 11 * 11
5  The compact factorization is: 3 * 11^2
6  4 nontrivial divisors: 3, 11, 33, 121
7
8  Enter an integer (an integer <= 1 will stop the program)
9  Your number: 3636
10 The factorization is: 2 * 2 * 3 * 3 * 101
11 The compact factorization is: 2^2 * 3^2 * 101
12 16 nontrivial divisors: 2, 3, 4, 6, 9, 12, 18, 36, 101, 202, 303,
      404, 606, 909, 1212, 1818
13
```

```
Enter an integer (an integer <= 1 will stop the program)
Your number: 76780
The factorization is: 2 * 2 * 5 * 11 * 349
The compact factorization is: 2^2 * 5 * 11 * 349
22 nontrivial divisors: 2, 4, 5, 10, 11, 20, 22, 44, 55, 110, 220,
   349, 698, 1396, 1745, 3490, 3839, 6980, 7678, 15356, 19195, 38390

Enter an integer (an integer <= 1 will stop the program)
Your number: 0
```

The empty lines are to clarify the prompt for receiving an input number.

2 Our Coding Plan

We will write a main method where the program interacts with the user to receive input numbers using a while or a do-while loop. We then write three methods for the three tasks, where each task receives an input integer and returns a `String` data.

In Part 1, we can use the following idea:

- Initialize a `StringBuilder` variable `builder` with the `String` literal `"The factorization is: "`.
- Initialize an `int` variable `need_to_factor` with `input`.
- Initialize an `int` variable `divisor` with `2`.
- While `need_to_factor > 1`, do the following;
  - While `divisor` divides `need_to_factor`, do the following:
    - Append `divisor` to the builder. The `StringBuilder` class automatically turns the integer to a `String` and appends to the builder's content.
      Before that, if `need_to_factor` has already been reduced from its initial value, `input`, append `" * "`.
    - Reduce `need_to_factor` by replacing it with `need_to_factor / divisor`
  - Increase the value of `divisor` by `1`.
- The factorization is complete, so return the `String` that the builder represents.

In Part 2, we can use the following idea:

- Initialize a `StringBuilder` variable `builder` with the `String` literal `"The compact factorization is: "`.
- Initialize an `int` variable `need_to_factor` with `input`.
- Initialize an `int` variable `divisor` with `2`.
- While `need_to_factor > 1`, do the following;

  - If `divisor` divides `need_to_factor`, do as follows:
    * Append `divisor` to the builder. The `StringBuilder` class makes an automatic conversion of the integer to a `String` data.
      Before that, if `need_to_factor` is less than its initial value, append `" * "`.
    * initialize an `int` variable `exponent` with the value of 0.
    * While `divisor` divides `need_to_factor`,
      · Reduce `need_to_factor` by replacing it with `need_to_factor / divisor`.
      · Increase the value of `exponent` by 1.
    * If `exponent > 1`, append `êxponent` to the builder.

- Increase `divisor` by `1`

In Part 3, the program iterates over a series of integers from `2` to `input - 1` using an iteration variable `divisor`. Like before, each discovered divisor is added to a `StringBuilder` variable `builder` initialized with the value of `" nontrivial divisors are found: "`. During the loop, for each divisor found, the program increases an integer variable representing the divisor count. The initial value of this count is 0. When the loop completes, the program inserts the value of the count at the beginning. Additionally, the program uses this count to determine if it must append `", "` before appending the divisor.