CSC120 2025S Lab No. 8
Reading Prime Factorization

This lab aims to write a program that receives, from the user, a `String` data representing a pair of prime power expressions separated by `"/"` and obtains a whose number expression representing the fraction.

Recall that the format of the prime factorization is:

```
r_1 * r_2 * ... * r_k
```

Here, `r_1, ..., r_k` are powers of prime. A power of prime is either a prime number of a prime number followed by a caret `^` or `**` and then by a positive integer. For example, `3`, `5^1`, and `7**2` are all prime powers while none of `-3`, `5^-1`, or `7+2`. Given a prime power, we call the prime that appears at the start the `base_integer` and the number after the caret the `exponent`. If a caret does come after `base_integer`, the `exponent` is 1. In `3`, `5^1`, and `7**2`, the `base_integer` is 3, 5, and 7, respectively, and the exponent is 1, 1, and 2, respectively.

Here are the rules about valid expressions:

- If a sequence of numerals is separated from another sequence immediately following that, there should be a star `*` or a caret `^`.
- A star `*` can appear only between two numeral sequences.
- A caret `^` can appear only between two numeral sequences.
- There shall be no characters in the sequence other than the whitespace, the star, the caret, and the numerals.
- Each base must be a prime number.
- The sequence of bases in the expression must be strictly increasing.

We envision to have the following four methods in the code:

1. `main()`: the method is responsible for receiving input, calling the method `convert()` with the input as the parameter, and reporting the return value; the method must be designed to receive any number of inputs and terminate with the input of CTRL-D.
2. receive an indefinite number of input

You write a method `convert()` that receives a `String` parameter `w`, which is supposed to encode a prime factorization and returns a `long` integer the prime factorization represents. The algorithm for `convert()` can be as follows:

1. Modify `w` as follows:
   (a) Replace each occurrence of `"**"` in it with `" "̂`.
   (b) Replace each occurrence of `"*"` in it with `" * "`.
   (c) Replace each occurrence of |””̂— in it with |” ”̂—.
   (d) Append `" *"` at the end.

   The modifications turn the input into a sequence of tokens readable using a `Scanner` object where numbers and non-numbers alternate.

2. Instantiate a `Scanner` object `in` with the updated `w` as the parameter.

3. Execute other initializations and declarations.
   (a) Declare a `long` variable `result` and Initialize it with the value of 1.
   (b) Declare `String` variable `operand`, an `int` variable, `exponent`, `base_integer`, and `prevBase`.
   (c) Initialize `prevBase` with the value of 1.

4. While there is a token remaining in `in`, do the following:
   (a) Obtain an `int` token and store it in `base_integer`.
   (b) Obtain the next `String` token and store it in `operand`.
   (c) If `operand` happens to be `"^"`, read the next token as an `int` token, store it in `exponent`, read the following `String` token, and store it in `operand`; otherwise, store the value of 1 in `exponent`.
   (d) Check to see if `base_integer` is a prime number, if `exponent` is greater than or equal to 1, `operand` is equal to `"*"`, and if `prevBase < base_integer`. Throw an `IllegalArgumentException` if any test fails.
   (e) Compute the value of `base_integer` raised to the power of `exponent`
   (f) Update `prevBase` with the value of `base`.

   Return the value of `result`.

Multiplying `result` by the `base` raised to the power of `exponent` can be done by updating the value of a `long` variable `result` by `base_integer` `exponent` times, where the initial value of `result` is 1.

Testing if `base_integer` is a prime can be done as follows: After initializing the value of an `int` variable `divisor` to `2`, and then while `divisor` is less than `base_integer` and `divisor` does not divide `base_integer` increase the value of `divisor` by 1. When the loop terminates, we can tell if the loop has stopped with the value of `divisor` less than `base_integer`. If that is the case, `base_integer` is composite; otherwise, it is prime.

In the method `main`, using a while-loop, repeat the process of prompting the user to enter an expression, receiving the expression using the `nextLine()` method, calling the conversion method, and reporting the result. The loop should terminate when the user presses CTRL-D.

Here is a sample program execution. After the user enters input, the second output line results from calling the optional method.

```
Enter an expression: 2^3 * 5
The number is 40.
Enter an expression: 2**3 * 5
The number is 40.
Enter an expression: 5*3
Exception in thread "main" java.lang.IllegalArgumentException: No increase in t
        at Compose.convert(Compose.java:99)
        at Compose.main(Compose.java:122)
```

The third one has a decreasing sequence of bases.

Here is another example.

```
Enter an expression: 11^11 * 13
The number is 3709051717943.
Enter an expression: ^D
```