Enhanced Postfix Evaluator

1. Problem Description

In class, we went over postfix expressions and an approach for evaluating them using a Stack. The Java code was likewise presented and is available in Canvas. You are to enhance this code and include the several additional operations listed below. Your Java program will continue to prompt the user for additional expressions until the user chooses to finish. Your solution should handle any exceptions gracefully and report the error.

1) Add the following new binary operators:
   - Modulus: e_1 e_2 %
   - Power (e_1 raised to the e_2 power): e_1 e_2 ^
   - Example Expressions:
     `17 5 %` (Result: 2)
     `3 4 ^` (Result: 81)

2) Add the following new unary operators:
   - Unary minus: e ~
   - Factorial: e !
   - Example Expressions:
     `12 ~` (Result: -12)
     `5 ~ ~ ~` (Result: -5)
     `6 !` (Result: 720)
     `3 ! 4 * 5 3 % - 4 2 ~ / *` (Result: -44)

3) Add the following relational, Boolean and ternary operators:
   - Relational Operators: e_1 e_2 >, e_1 e_2 <, e_1 e_2 =
   - Boolean Operators: e_1 e_2 &, e_1 e_2 |
   - Ternary Conditional Operator (if e_1 is true, value is e_2, otherwise e_3): e_1 e_2 e_3 ?
   - Note: For Boolean values, use the same convention that the C language does: 0 is false, non-0 is true.
   - Example Expressions:
     `5 3 >` (Result: 1)
     `5 3 <` (Result: 0)
     `5 5 =` (Result: 1)
     `10 5 &` (Result: 1) Anything non-0 is considered "true"
     `5 3 > 2 3 > |` (Result: 1)
     `0 0 |` (Result: 0)
     `5 3 < 2 5 = 1 7 < | |` `10 20 ?` (Result: 10)
     `1 2 ?` (Result: Error)

2. Notes

- Do not create these classes in a package. For Eclipse, this means using the *default* package.
- Turn in only your Java files: PostfixTester.java and PostfixEvaluator.java.
- In Java the ^ operator is *exclusive-OR*, not power, as we are using it here.

3. Required Main Class

PostfixTester

4. Required Input

One or more postfix expressions, with ‘y’ or ‘n’ to continue

5. Required Output

Your output should look like the following example. It must include your name. Note, in the example below the repetitive `Evaluate another expression [Y/N]?` has been removed after the first example. However, your program should ask every time. User input is shown in GREEN.

```
Postfix Expression Evaluator – Your Name

Post-fix expression: 3 4 * 2 5 + - 4 2 / *
   Result = 10

Evaluate another expression [Y/N]? Y

Post-fix expression: 17 5 %
   Result = 2

Post-fix expression: 3 4 ^
   Result = 81

Post-fix expression: 5 +
   ERROR: Insufficient operands for +.

Post-fix expression: 1 2 3 +
   ERROR: 1 too few operators.

Post-fix expression: 12 ~
   Result = -12

Post-fix expression: 5 ~ ~ ~
   Result = -5

Post-fix expression: 6 !
   Result = 720

Post-fix expression: 3 ! 4 * 5 3 % - 4 2 ~ / *
   Result = -44

Post-fix expression: 5 3 >
   Result = 1

Post-fix expression: 5 3 <
   Result = 0

Post-fix expression: 5 5 =
   Result = 1

Post-fix expression: 10 5 &
   Result = 1

Post-fix expression: 5 3 > 2 3 > |
   Result = 1

Post-fix expression: 0 0 |
   Result = 0

Post-fix expression: 5 3 < 2 5 = 1 7 < | | 10 20 ?
   Result = 10

Post-fix expression: 1 2 ?
   ERROR: Insufficient operands for ?.
```