CSCI 3320: Data Structures
HW 2: Implementing a Calculator
Due: 10/11 (Wed) 11:59 pm (8 points)

1. Project Details

In this project, you will implement a calculator using a stack. The calculator will get input (infix notation) from a user, show the postfix notation, and print the result. That is, you need to convert the infix notation into the postfix notation and evaluate it. If the input is not valid, e.g., "1 + 3*", the error message will be printed. The calculator will support five operators: +, -, *, /, and ^.
The calculator will receive the inputs continuously until it is terminated. You can assume that operands are only numbers.

```
E.g.,> Python Calculator.py or Java Calculator
> input: 1+4
postfix: 14+
result: 5

> input: 9+2+
result: invalid inputs

> input: A+B
result: only numbers are available.

> input: (5 + 9) * ((2 – 4 * 2) / 2) + 2^9
postfix: 5 9 + 2 4 2 * - 2 / * 2 9 ^ +
result: 470

> input: …
```

2. Implementation Details

A skeleton code (**Calculator.py and Calcuator.java**) will be provided. Please add comments to your code for the instructor to understand your code. You must use a stack to complete this project and you can use a built-in List in Python which provides stack operations. For Java code, you can use java.util.Stack. You can use the code from the slides and find related information from the Internet or books.

Implementation details and submission requirements:

1. You must implement the `convert ()` function based on the pseudo-code provided in the skeleton code. For the `evaluate ()` function, you must follow the instructions on the skeleton code. For both functions, you must use the **stack**.
2. Your code must align with the pseudo-code and instructions. Please show your code with the corresponding pseudo-code and instructions. For example, your source in Python will look as follows.
3.
```
#convert ()
#if c == '(' stack.push (c)     # given pseudo code
if c == '(':                    # your code.
    self.stack.push (c)
…
#evaluate ()
for c in postfix                #• Repeat
    if c == '+' or '-' … :      #• If operand, push onto stack
…
```
4. <u>You are not allowed</u> to add (create) additional functions e.g., `precedence ()`, `isOperator ()`, etc., that you can find on the Internet.
5. <u>You are not allowed</u> to create variables using a list or a dictionary like below.
```
priority = {'+': 1, '-': 1, '*': 2, '/': 2, '^': 3}
operators = ['(', ')', '-', '+', '*', '/', '^']
```
6. The `evaluate ()` function must support **double data type** to support division.
7. You must create and make your own test cases (various inputs) to fully test your calculator. You will lose points if there are any exceptions (crashes) while testing your program. For example, I will use some test cases as inputs as follows,
```
> 1+4
> (5 + 9) * ((2 – 4 * 2) / 2) + 2^9
> 5/2*5/2+1
> A+B
> (5+4 *
> (1 + 2) * 3 * ((6 * 6) + 1)
```
8. You can discuss this project with your classmates to get some hints. However, **it is not allowed to 1) share the solution with your classmates, 2) post this project to websites e.g., chegg.com to get solutions, and 3) just copy and paste the code you found on the Internet or other references.** In short, **<u>please complete the project by yourself</u>**. **I will use tools to detect cheating (copy and paste).** The exam will ask you how you implement these functions.

Extra credit: If your program supports multiple-digit numbers, e.g., 200, 1000, and 150, you will receive 1 extra credit.

3. Deliverables

Please submit your code on Canvas by the due date. Please put your information in the code.