/\* 4. Evaluate a postfix expression using a stack.

• Test Case 1:

Input: "5 3 + 2 \*"

Output: 16

• Test Case 2:

Input: "4 5 \* 6 /"

Output: 3 \*/

import java.util.Scanner;

public class question4 {

private int[] stack;

private int top;

private int maxSize;

public question4(int size) {

stack = new int[size];

maxSize = size;

top = -1;

}

public void push(int value) {

if (top < maxSize - 1) {

stack[++top] = value;

}

}

public int pop() {

if (top >= 0) {

return stack[top--];

}

return -1; // Indicates stack underflow

}

public static int evaluate(String expression) {

String[] tokens = expression.split(" ");

question4 stack = new question4(tokens.length);

for (String token : tokens) {

if (token.matches("-?\\d+")) { // Check if token is an integer

stack.push(Integer.parseInt(token));

} else {

int b = stack.pop();

int a = stack.pop();

switch (token) {

case "+":

stack.push(a + b);

break;

case "-":

stack.push(a - b);

break;

case "\*":

stack.push(a \* b);

break;

case "/":

stack.push(a / b);

break;

}

}

}

return stack.pop();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Take input from user

System.out.print("Enter postfix expression: ");

String expression = scanner.nextLine();

// Evaluate the postfix expression

int result = evaluate(expression);

System.out.println("Input: \"" + expression + "\"");

System.out.println("Output: " + result);

}

}  
  
explanation  
The program evaluates postfix expressions using a custom stack implementation, where the user inputs the expression, and the program processes it by pushing numbers onto the stack and applying operators by popping the required operands from the stack. The PostfixEvaluator class manages the stack operations with an array, ensuring no external frameworks are used. The evaluate method parses the expression, handles numbers and operators, and performs arithmetic operations accordingly. The final result is the last remaining value on the stack, which is then printed. The time complexity is O(n) as each token is processed once, and the space complexity is O(n) due to the stack storing all tokens in the worst case.  
  
output  
