To implement a program that can take an expression in postfix notation and output its value, we will follow these steps:

1. **Input**: Read the postfix expression (which can be space-separated).
2. **Process the expression**: Use a stack to evaluate the expression.
3. **Output**: Print the result of the evaluated postfix expression.

**Steps for Evaluating Postfix Expression:**

* If the token is a number, push it onto the stack.
* If the token is an operator, pop the top two elements from the stack, apply the operator, and push the result back onto the stack.
* After processing all tokens, the result will be the only element left in the stack.

**Example Expression:**

For the expression 5 2 + 8 3 - \* 4 /, the result should be 8.75.

**Java Program to Evaluate Postfix Expression:**

import java.util.Stack;

public class PostfixEvaluator {

// Method to evaluate the postfix expression

public static double evaluatePostfix(String expression) {

// Stack to hold the operands

Stack<Double> stack = new Stack<>();

// Split the expression into tokens (space-separated)

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

// Iterate through each token in the expression

for (String token : tokens) {

if (isOperator(token)) {

// If token is an operator, pop two operands and apply the operator

double operand2 = stack.pop();

double operand1 = stack.pop();

double result = applyOperator(operand1, operand2, token);

stack.push(result);

} else {

// If the token is a number, push it to the stack

stack.push(Double.parseDouble(token));

}

}

// The result is the only value left in the stack

return stack.pop();

}

// Method to check if a string is an operator

private static boolean isOperator(String token) {

return token.equals("+") || token.equals("-") || token.equals("\*") || token.equals("/");

}

// Method to apply the operator to the operands

private static double applyOperator(double operand1, double operand2, String operator) {

switch (operator) {

case "+":

return operand1 + operand2;

case "-":

return operand1 - operand2;

case "\*":

return operand1 \* operand2;

case "/":

return operand1 / operand2;

default:

throw new IllegalArgumentException("Unknown operator: " + operator);

}

}

public static void main(String[] args) {

// Input the postfix expression

System.out.println("Enter a postfix expression (space-separated): ");

java.util.Scanner scanner = new java.util.Scanner(System.in);

String expression = scanner.nextLine();

// Evaluate the expression

double result = evaluatePostfix(expression);

// Output the result

System.out.println("The result of the postfix expression is: " + result);

scanner.close();

}

}

**Explanation:**

1. **evaluatePostfix(String expression)**:
   * This method takes a postfix expression as input, splits it into tokens (space-separated), and processes each token.
   * For each token:
     + If it is a number, it is pushed onto the stack.
     + If it is an operator, the two operands are popped from the stack, the operator is applied, and the result is pushed back onto the stack.
   * At the end of the process, the result will be the only item remaining on the stack, which is then returned.
2. **isOperator(String token)**:
   * This method checks if the token is an operator (+, -, \*, or /).
3. **applyOperator(double operand1, double operand2, String operator)**:
   * This method applies the operator to the two operands and returns the result.
4. **Main Method**:
   * Reads the postfix expression from the user.
   * Calls the evaluatePostfix() method to evaluate the expression.
   * Prints the result of the evaluated expression.

**Example Usage:**

**Input:**

Enter a postfix expression (space-separated):

5 2 + 8 3 - \* 4 /

**Output:**The result of the postfix expression is: 8.75

**Detailed Explanation of the Example:**

For the postfix expression 5 2 + 8 3 - \* 4 /, the steps are as follows:

1. Push 5 → Stack: [5]
2. Push 2 → Stack: [5, 2]
3. Apply + → Pop 2 and 5, compute 5 + 2 = 7, and push 7 → Stack: [7]
4. Push 8 → Stack: [7, 8]
5. Push 3 → Stack: [7, 8, 3]
6. Apply - → Pop 3 and 8, compute 8 - 3 = 5, and push 5 → Stack: [7, 5]
7. Apply \* → Pop 5 and 7, compute 7 \* 5 = 35, and push 35 → Stack: [35]
8. Push 4 → Stack: [35, 4]
9. Apply / → Pop 4 and 35, compute 35 / 4 = 8.75, and push 8.75 → Stack: [8.75]
10. The final result is 8.75.

**Time Complexity:**

* The time complexity is **O(n)**, where n is the number of tokens in the expression. Each token is processed once, and the stack operations (push, pop) are constant time operations.