1. Implement a method with signature transfer(S, T) that transfers all elements from stack S onto stack T, so that the element that starts at the top of S is the first to be inserted onto T, and the element at the bottom of S ends up at the top of T.

public static <T> void transfer(Stack<T> S, Stack<T> T) {

Stack<T> tempStack = new Stack<>();

// Transfer elements from stack S to tempStack in reverse order

while (!S.isEmpty()) {

tempStack.push(S.pop());

}

// Transfer elements from tempStack to stack T in the desired order

while (!tempStack.isEmpty()) {

T.push(tempStack.pop());

}

}

2.Give a recursive method for removing all the elements from a stack.

public static <T> void removeAll(Stack<T> stack) {

if (!stack.isEmpty()) {

stack.pop();

removeAll(stack);

}

}

3. Postfix notation is an unambiguous way of writing an arithmetic expression without parentheses. It is defined so that if “(exp1)op(exp2)” is a normal fully parenthesized expression whose operation is op, the postfix version of this is “pexp1 pexp2 op”, where pexp1 is the postfix version of exp1 and pexp2 is the postfix version of exp2. The postfix version of a single number or variable is just that number or variable. So, for example, the postfix version of “((5 + 2) ∗ (8 − 3))/4” is “5 2 + 8 3 − ∗ 4 /”. Describe a nonrecursive way of evaluating an expression in postfix notation.

import java.util.Stack;

public class PostfixEvaluator {

public static double evaluate(String postfixExpression) {

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

String[] tokens = postfixExpression.split("\\s+");

for (String token : tokens) {

if (isOperand(token)) {

double operand = Double.parseDouble(token);

stack.push(operand);

} else if (isOperator(token)) {

double operand2 = stack.pop();

double operand1 = stack.pop();

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

stack.push(result);

}

}

return stack.pop();

}

private static boolean isOperand(String token) {

try {

Double.parseDouble(token);

return true;

} catch (NumberFormatException e) {

return false;

}

}

private static boolean isOperator(String token) {

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

}

private static double performOperation(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("Invalid operator: " + operator);

}

}

}

4.Implement the clone( ) method for the ArrayStack class.  
public ArrayStack<E> clone() {

ArrayStack<E> clonedStack = new ArrayStack<>(size);

clonedStack.elements = Arrays.copyOf(elements, size);

clonedStack.size = size;

return clonedStack;

}

5. Implement a program that can input an expression in postfix notation (see Exercise C-6.19) and output its value

import java.util.Scanner;

public class PostfixCalculator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String postfixExpression = scanner.nextLine();

double result = PostfixEvaluator.evaluate(postfixExpression);

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

}