Topics

* Create Stack Interface
* Create Stack Using Array
* Create Stack Using Linked Lists
* Implement Basic Methods of Stack
* isEmpty()
* size()
* top()
* push(E e)
* pop()

Homework

* 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 <E> void transfer(Stack<E> S, Stack<E> T) {

while (!S.isEmpty()) {

T.push(S.pop());

}

}

public static <E> void clearStack(Stack<E> S) {

if (!S.isEmpty()) {

S.pop(); // حذف العنصر العلوي

clearStack(S); // استدعاء نفسه لحذف باقي العناصر

}

}

----------------------------------------------------------------------------------------

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

public static <E> void clearStack(Stack<E> S) {

if (!S.isEmpty()) {

S.pop(); // حذف العنصر العلوي

clearStack(S); // استدعاء نفسه لحذف باقي العناصر

}

}

---------------------------------------------------------------------------------------------

* 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 int evaluatePostfix(String expression) {

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

for (String token : expression.split(" ")) {

if (token.matches("-?\\d+")) { // التحقق إن كان رقمًا

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

} else {

int b = stack.pop(); // العنصر الثاني

int a = stack.pop(); // العنصر الأول

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(); // النتيجة النهائية

}

public static void main(String[] args) {

String postfix = "5 2 + 8 3 - \* 4 /";

System.out.println("Result: " + evaluatePostfix(postfix)); // الناتج: 7

}

}  
---------------------------------------------------------------------------------------

* Implement the clone( ) method for the ArrayStack class.

public class ArrayStack<E> implements Cloneable {

private E[] data;

private int top = -1;

@SuppressWarnings("unchecked")

public ArrayStack(int capacity) {

data = (E[]) new Object[capacity];

}

public void push(E e) { data[++top] = e; }

public E pop() { return data[top--]; }

public boolean isEmpty() { return top == -1; }

@Override

public ArrayStack<E> clone() {

try {

ArrayStack<E> cloned = (ArrayStack<E>) super.clone();

cloned.data = data.clone(); // نسخ المصفوفة

return cloned;

} catch (CloneNotSupportedException e) {

throw new AssertionError(); // لن يحدث ذلك

}

}

}

----------------------------------------------------------------------------------------

* 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 a postfix expression: ");

String postfix = scanner.nextLine();

System.out.println("Result: " + PostfixEvaluator.evaluatePostfix(postfix));

}

}