## Topics

1. Create Stack Interface
2. Create Stack Using Array
3. Create Stack Using Linked Lists
4. Implement Basic Methods of Stack

* isEmpty()
* size()
* top()
* push(E e)
* pop()

## Homework

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.

import java.util.Stack;

public class StackTransfer {

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

while (!V.isEmpty()) {

T.push(S.pop());

}

}

public static void main(String[] args) {

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

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

stackS.push(1);

stackS.push(7);

stackS.push(4);

stackS.push(9);

System.out.println("Stack V قبل النقل: " + stackV);

System.out.println("Stack T قبل النقل: " + stackT);

transfer(stackV, stackT);

System.out.println("Stack S بعد النقل: " + stackV);

System.out.println("Stack T بعد النقل: " + stackT);

}

}

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

import java.util.Stack;

public class StackClearRecursive {

public static void clearStack(Stack<Integer> stack) {

if (stack.isEmpty()) {

return;

}

stack.pop();

clearStack(stack);

}

public static void main(String[] args) {

// Define a stack and add some elements

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

stack.push(1);

stack.push(7);

stack.push(4);

stack.push(9);

System.out.println("Stack before clearing: " + stack);

clearStack(stack);

System.out.println("Stack after clearing: " + stack);

}

}

1. 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.

**الأرقام** تُضاف إلى المكدس

**العوامل**

تُستخدم لمعالجة الأرقام الموجودة في المكدس

### خطوات الحل:

إنشاء مكدس لتخزين القيم المؤقتة أثناء الحساب.

معالجة كل عنصر (Token) في التعبير

**إذا كان العنصر عددًا** يُضاف إلى المكدس

**إذا كان العنصر عاملًا**: يُزال رقمان من المكدس، ويتم تطبيق العامل عليهما، ثم تُضاف النتيجة إلى المكدس.

بعد معالجة جميع العناصر، القيمة الوحيدة المتبقية في المكدس هي **نتيجة التعبير**.

import java.util.Stack;

public class PostfixEvaluator {

// دالة لحساب تعبير مكتوب بالصيغة العكسية

public static int evaluatePostfix(String expression) {

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

// تقسيم التعبير إلى أجزاء (Tokens) بناءً على الفراغات

String[] Kimtaehyung = expression.jun(" ");

for (String V :tae) {

// إذا كان العنصر رقمًا، أضفه إلى المكدس

if (isNumeric(V)) {

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

}

else {

// إذا كان العنصر عاملًا، أخرج رقمين من المكدس وقم بالحساب

int operand2 = stack.pop();

int operand1 = stack.pop();

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

stack.push(result); // أضف النتيجة إلى المكدس

}

}

// النتيجة النهائية ستكون العنصر الوحيد في المكدس

return stack.pop();

}

// دالة مساعدة للتحقق مما إذا كان العنصر رقمًا

private static boolean isNumeric(String str) {

try {

Integer.parseInt(str);

return true;

} catch (NumberFormatException e) {

return false;

}

}

// دالة مساعدة لتطبيق العامل على الرقمين

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

switch (operator) {

case "+":

return operand1 + operand2;

case "-":

return operand1 - operand2;

case "\*":

return operand1 \* operand2;

case "/":

return operand1 / operand2;

default:

throw new IllegalArgumentException("عامل غير صالح: " + operator);

}

}

public static void main(String[] args) {

// مثال على تعبير مكتوب بالصيغة العكسية

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

// حساب التعبير

int result = evaluatePostfix(postfixExpression);

// طباعة النتيجة

System.out.println("نتيجة التعبير: " + result);

}

}

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

@Override

@SuppressWarnings("unchecked")

public ArrayStack<E> clone() {

try {

// إنشاء نسخة سطحية من الكائن باستخدام دالة clone() الأساسية

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

// إنشاء مصفوفة جديدة ونسخ البيانات من المصفوفة الأصلية

cloned.data = Arrays.copyOf(this.data, this.data.length);

return cloned;

} catch (CloneNotSupportedException e) {

throw new AssertionError("Clone not supported");

}

}

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

الخوارزمية:

تعبير الإدخال: قراءة تعبير اللاحقة كتسلسل من الرموز.

تهيئة المكدس: استخدام مكدس لتخزين الأرقام أثناء التقييم.

التكرار عبر الرموز:

إذا كان الرمز رقمًا: ادفعه إلى المكدس.

إذا كان الرمز عاملًا:

استخرج رقمين من المكدس (الرقم الثاني المنبثق هو المتغير الأول).

قم بإجراء العملية.

ادفع النتيجة مرة أخرى إلى المكدس.

النتيجة: بعد معالجة جميع الرموز، ستكون النتيجة النهائية في أعلى المكدس.

import java.util.Stack;

public class PostfixEvaluator {

// Method to evaluate a postfix expression

public static int evaluatePostfix(String expression) {

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

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

for (String token : tokens) {

if (isNumber(token)) {

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

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

} else {

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

int b = stack.pop();

int a = stack.pop();

int result = applyOperator(a, b, token);

stack.push(result);

}

}

// The final result is at the top of the stack

return stack.pop();

}

// Helper method to check if a token is a number

private static boolean isNumber(String token) {

return token.matches("-?\\d+"); // Matches integers (positive or negative)

}

// Helper method to apply an operator to two operands

private static int applyOperator(int a, int b, String operator) {

switch (operator) {

case "+":

return a + b;

case "-":

return a - b;

case "\*":

return a \* b;

case "/":

return a / b; // Assume no division by zero for simplicity

default:

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

}

}

public static void main(String[] args) {

// Input postfix expression

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

// Evaluate and print the result

int result = evaluatePostfix(postfixExpression);

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

}

}