## 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 {

// دالة لنقل العناصر من المكدس S إلى المكدس T

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

while (!S.isEmpty()) {

T.push(S.pop());

}

}

public static void main(String[] args) {

// إنشاء مكدسين S و T

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

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

// إضافة عناصر إلى المكدس S

S.push(1);

S.push(2);

S.push(3);

S.push(4);

S.push(5); // العنصر 5 في الأعلى

// طباعة المكدسين قبل النقل

System.out.println("قبل النقل:");

System.out.println("S: " + S);

System.out.println("T: " + T);

// تنفيذ النقل

transfer(S, T);

// طباعة المكدسين بعد النقل

System.out.println("\nبعد النقل:");

System.out.println("S: " + S);

System.out.println("T: " + T);

}

}

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

**تحليل الحل**

1. **المكدس يعمل وفق مبدأ LIFO (Last In, First Out)**، أي أن آخر عنصر يُضاف هو أول عنصر يتم حذفه عند استخدام pop().
2. **الاستدعاء الذاتي (Recursion)** يعتمد على تقسيم المشكلة إلى مشكلة أصغر، لذا:
   * إذا كان المكدس فارغًا (stack.isEmpty())، فإننا ننهي الاستدعاء.
   * إذا كان هناك عناصر في المكدس:
     + نحذف العنصر العلوي باستخدام pop().
     + نستدعي الدالة نفسها على المكدس المتبقي حتى يصبح فارغًا تمامًا.

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.

import java.util.Stack;

public class StackRecursiveRemove {

// دالة لحذف جميع العناصر باستخدام الاستدعاء الذاتي (Recursion)

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

// حالة التوقف: إذا كان المكدس فارغًا، نوقف الاستدعاء

if (stack.isEmpty()) {

return;

}

// إزالة العنصر العلوي

stack.pop();

// استدعاء الدالة نفسها على بقية العناصر

recursiveRemove(stack);

}

public static void main(String[] args) {

// إنشاء مكدس وإضافة عناصر إليه

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

stack.push(1);

stack.push(2);

stack.push(3);

stack.push(4);

stack.push(5); // العنصر 5 هو العلوي

// طباعة المكدس قبل الحذف

System.out.println("قبل الحذف: " + stack);

// استدعاء الدالة لحذف جميع العناصر

recursiveRemove(stack);

// طباعة المكدس بعد الحذف

System.out.println("بعد الحذف: " + stack);

}

}

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

**تحليل الحل**

في Java، عند نسخ كائن باستخدام =، يتم نسخ **الإشارة (reference)** فقط، وليس البيانات الفعلية. لذلك، لإنشاء نسخة حقيقية (Deep Copy)، نحتاج إلى:

1. **إنشاء كائن جديد من ArrayStack**.
2. **نسخ جميع العناصر من الكائن الأصلي إلى النسخة الجديدة**.

**أنواع النسخ في Java**

* **النسخ السطحي (Shallow Copy):** يتم نسخ المصفوفة ولكنها تبقى تشير إلى نفس العناصر في الذاكرة.
* **النسخ العميق (Deep Copy):** يتم إنشاء نسخة جديدة من المصفوفة، بحيث يصبح المكدس المستنسخ مستقلاً تمامًا عن الأصل.

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

في **الترميز اللاحق (Postfix)**:

* **لا تحتاج الأقواس** لأن العمليات الحسابية تُنفّذ حسب ترتيب الإدخال.
* يتم وضع **المعاملات (Operands) قبل العمليات الحسابية (Operators)**.
* كلما صادفنا **عملية حسابية (مثل +, -, \*, /)**، نأخذ آخر **رقمين من المكدس**، وننفّذ العملية بينهما، ثم نعيد النتيجة إلى المكدس.

import java.util.Stack;

import java.util.Scanner;

public class PostfixEvaluator {

// دالة لحساب قيمة تعبير postfix باستخدام مكدس

public static int evaluatePostfix(String expression) {

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

// تقسيم التعبير إلى أجزاء (Tokens) باستخدام المسافات

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

for (String token : tokens) {

if (isNumeric(token)) {

// إذا كان رقماً، ندخله إلى المكدس

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

} else {

// إذا كان عملية حسابية، نخرج آخر رقمين من المكدس

int operand2 = stack.pop(); // الرقم الثاني (يُخرج أولًا)

int operand1 = stack.pop(); // الرقم الأول

int result;

// تنفيذ العملية المناسبة

switch (token) {

case "+":

result = operand1 + operand2;

break;

case "-":

result = operand1 - operand2;

break;

case "\*":

result = operand1 \* operand2;

break;

case "/":

if (operand2 == 0) {

throw new ArithmeticException("خطأ: القسمة على صفر غير مسموحة!");

}

result = operand1 / operand2;

break;

default:

throw new IllegalArgumentException("عملية غير صالحة: " + token);

}

// تخزين النتيجة في المكدس

stack.push(result);

}

}

// النتيجة النهائية تكون في قمة المكدس

return stack.pop();

}

// دالة للتحقق مما إذا كانت سلسلة نصية تمثل رقمًا

private static boolean isNumeric(String str) {

try {

Integer.parseInt(str);

return true;

} catch (NumberFormatException e) {

return false;

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// إدخال تعبير Postfix من المستخدم

System.out.print("أدخل تعبير postfix: ");

String expression = scanner.nextLine();

try {

int result = evaluatePostfix(expression);

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

} catch (Exception e) {

System.out.println("حدث خطأ: " + e.getMessage());

}

scanner.close();

}

}