## 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()

// تعريف واجهة Stack

public interface Stack<E> {

boolean isEmpty(); // التحقق إذا كان المكدس فارغًا

int size(); // إرجاع عدد العناصر في المكدس

E top(); // عرض العنصر الموجود في قمة المكدس دون إزالته

void push(E element); // إضافة عنصر إلى المكدس

E pop(); // إزالة العنصر الموجود في قمة المكدس

}

// تنفيذ المكدس باستخدام مصفوفة

public class ArrayStack<E> implements Stack<E> {

private static final int DEFAULT\_CAPACITY = 10; // السعة الافتراضية

private E[] data; // مصفوفة لتخزين العناصر

private int top = -1; // مؤشر لقمة المكدس

@SuppressWarnings("unchecked")

public ArrayStack(int capacity) {

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

}

public ArrayStack() {

this(DEFAULT\_CAPACITY);

}

@Override

public boolean isEmpty() {

return top == -1;

}

@Override

public int size() {

return top + 1;

}

@Override

public E top() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty");

}

return data[top];

}

@Override

public void push(E element) {

if (size() == data.length) {

throw new IllegalStateException("Stack is full");

}

data[++top] = element;

}

@Override

public E pop() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty");

}

E result = data[top];

data[top--] = null; // إزالة المرجع لتجنب تسريب الذاكرة

return result;

}

}

// تنفيذ المكدس باستخدام قائمة مرتبطة

public class LinkedStack<E> implements Stack<E> {

// تعريف عقدة قائمة مرتبطة

private static class Node<E> {

private E element;

private Node<E> next;

public Node(E element, Node<E> next) {

this.element = element;

this.next = next;

}

}

private Node<E> top = null; // قمة المكدس

private int size = 0; // عدد العناصر في المكدس

@Override

public boolean isEmpty() {

return size == 0;

}

@Override

public int size() {

return size;

}

@Override

public E top() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty");

}

return top.element;

}

@Override

public void push(E element) {

top = new Node<>(element, top);

size++;

}

@Override

public E pop() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty");

}

E result = top.element;

top = top.next;

size--;

return result;

}

}

// برنامج اختبار المكدس

public class StackTest {

public static void main(String[] args) {

// اختبار المكدس باستخدام المصفوفة

System.out.println("Testing ArrayStack:");

Stack<Integer> arrayStack = new ArrayStack<>();

arrayStack.push(10);

arrayStack.push(20);

arrayStack.push(30);

System.out.println("Top: " + arrayStack.top()); // Output: 30

System.out.println("Size: " + arrayStack.size()); // Output: 3

System.out.println("Pop: " + arrayStack.pop()); // Output: 30

System.out.println("Size after pop: " + arrayStack.size()); // Output: 2

// اختبار المكدس باستخدام القائمة المرتبطة

System.out.println("\nTesting LinkedStack:");

Stack<String> linkedStack = new LinkedStack<>();

linkedStack.push("A");

linkedStack.push("B");

linkedStack.push("C");

System.out.println("Top: " + linkedStack.top()); // Output: C

System.out.println("Size: " + linkedStack.size()); // Output: 3

System.out.println("Pop: " + linkedStack.pop()); // Output: C

System.out.println("Size after pop: " + linkedStack.size()); // Output: 2

}

}

## 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.

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

while (!S.isEmpty()) {

T.push(S.pop());

}

}

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

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

if (stack.isEmpty()) {

return;

}

stack.pop(); // إزالة العنصر من القمة

removeAll(stack); // استدعاء ذاتي لإزالة باقي العناصر

}

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.

public static int evaluatePostfix(String expression) {

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

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

for (String token : tokens) {

if (isNumeric(token)) { // إذا كان الرقم

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

} else { // إذا كان العملية الحسابية

int b = stack.pop();

int a = stack.pop();

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();

}

private static boolean isNumeric(String str) {

return str.matches("-?\\d+(\\.\\d+)?");

}

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

public class ArrayStack<E> implements Cloneable {

private E[] data;

private int size;

// Constructor and other methods...

@Override

public ArrayStack<E> clone() {

try {

@SuppressWarnings("unchecked")

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

cloned.data = data.clone(); // نسخ مصفوفة البيانات

return cloned;

} catch (CloneNotSupportedException e) {

throw new AssertionError();

}

}

}

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

import java.util.Stack;

public class PostfixEvaluator {

public static void main(String[] args) {

String expression = "5 2 + 8 3 - \* 4 /"; // مثال على تعبير Postfix

int result = evaluatePostfix(expression);

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

}

public static int evaluatePostfix(String expression) {

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

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

for (String token : tokens) {

if (isNumeric(token)) { // إذا كان الرقم

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

} else { // إذا كان العملية الحسابية

int b = stack.pop();

int a = stack.pop();

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();

}

private static boolean isNumeric(String str) {

return str.matches("-?\\d+(\\.\\d+)?");

}

}