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()

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.
2. Give a recursive method for removing all the elements from a 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.
4. Implement the clone( ) method for the ArrayStack class.
5. Implement a program that can input an expression in postfix notation (see Exercise C-6.19) and output its value
6. إنشاء واجهة Stack:

public interface Stack<E> {

boolean isEmpty();

int size();

E top();

void push(E element);

E pop();

}

يتم تعريف واجهة Stack بواجهات الطرق الأساسية للعمليات في الدفع والاستخراج من العناصر في الستاك.

1. إنشاء ستاك باستخدام المصفوفة:

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

private static final int DEFAULT\_CAPACITY = 10;

private E[] data;

private int topIndex;

public ArrayStack() {

this(DEFAULT\_CAPACITY);

}

public ArrayStack(int capacity) {

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

topIndex = -1;

}

public boolean isEmpty() {

return topIndex == -1;

}

public int size() {

return topIndex + 1;

}

public E top() {

if (isEmpty()) {

return null;

}

return data[topIndex];

}

public void push(E element) {

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

resize(2 \* data.length);

}

topIndex++;

data[topIndex] = element;

}

public E pop() {

if (isEmpty()) {

return null;

}

E poppedElement = data[topIndex];

data[topIndex] = null;

topIndex--;

if (size() > 0 && size() == data.length / 4) {

resize(data.length / 2);

}

return poppedElement;

}

private void resize(int capacity) {

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

for (int i = 0; i <= topIndex; i++) {

newData[i] = data[i];

}

data = newData;

}

}

تم تنفيذ ستاك باستخدام المصفوفة. يتم استخدام مصفوفة لتخزين العناصر ومؤشر topIndex لتتبع العنصر الأعلى في الستاك.

1. إنشاء ستاك باستخدام القوائم المرتبطة:

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

private SinglyLinkedList<E> list;

public LinkedStack() {

list = new SinglyLinkedList<>();

}

public boolean isEmpty() {

return list.isEmpty();

}

public int size() {

return list.size();

}

public E top() {

return list.first();

}

public void push(E element) {

list.addFirst(element);

}

public E pop() {

return list.removeFirst();

}

}

تم تنفيذ ستاك باستخدام القوائم المرتبطة. يتم استخدام قائمة مرتبطة لتخزين العناصر، ويتم تنفيذ العمليات الأساسية للستاك باستخدام طرق القائمة المرتبطة.

1. تنفيذ الطرق الأساسية للستاك:

public interface Stack<E> {

// الطرق الأساسية الأخرى

boolean isEmpty();

int size();

E top();

void push(E element);

E pop();

}

الطرق المنفذة تشمل:

* isEmpty(): تحقق مما إذا كان الستاك فارغًا أم لا.
* size(): تعيد حجم الستاك.
* top(): تعيد العنصر الأعلى في الستاك.
* push(E element): تقوم بإضافة عنصر إلى الستاك.
* pop(): تقوم بإزالة وإرجاع العنصر الأعلى في الستاك.

1. تنفيذ الطريقة transfer(S, T) التي تنقل جميع العناصر من ستاك S إلى ستاك T بحيث يتم إدراج العنصر الذي يبدأ من الأعلى في S أولاً في T، وينتهي العنصر في أسفل S في الأعلى في T:

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

while (!S.isEmpty()) {

T.push(S.pop());

}

}

يتم استخدام حلقة while لنقل العناصر من ستاك S إلى ستاك T بواسطة دالة الدفع والاستخراج.

1. إعطاء طريقة متكررة لإزالة جميع العناصر من ستاك:

public void removeAll(Stack<E> stack) {

if (!stack.isEmpty()) {

stack.pop();

removeAll(stack);

}

}

تستخدم الطريقة نفسها بشكل متكرر لإزالة العناصر وحتى يتم تفريغ الستاك.

1. وصف طريقة غير متكررة لتقييم تعبير في العلامة اللاحقة:

public int evaluatePostfix(String postfixExpression) {

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

for (int i = 0; i < postfixExpression.length(); i++) {

char ch = postfixExpression.charAt(i);

if (Character.isDigit(ch)) {

stack.push(Character.getNumericValue(ch));

} else {

int operand2 = stack.pop();

int operand1 = stack.pop();

switch (ch) {

case '+':

stack.push(operand1 + operand2);

break;

case '-':

stack.push(operand1 - operand2);

break;

case '\*':

stack.push(operand1 \* operand2);

break;

case '/':

stack.push(operand1 / operand2);

break;

}

}

}

return stack.pop();

}

تستخدم الطريقة الستاك لتقييم التعبير الذي تم تمريره في العلامة اللاحقة. تمشي الطريقة عبر التعبير حرفًا تلو الآخر. إذا كان الحرف عبارة عن رقم، يتم دفعه في الستاك. إذا كان الحرف عبارة عن عملية، يتم سحب قيمتين من الستاك وتنفيذ العملية المناسبة ثم يتم دفع النتيجة النهائية في الستاك. في النهاية، يتم إرجاع القيمة العلوية في الستاك كنتيجة التقييم.

1. تنفيذ طريقة clone() لفئة ArrayStack:

@Override

public ArrayStack<E> clone() {

ArrayStack<E> clonedStack = new ArrayStack<>(data.length);

clonedStack.topIndex = this.topIndex;

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

return clonedStack;

}

الطريقة clone() تقوم بإنشاء نسخة من الستاك بنفس الحجم والعناصر. يتم نسخ العناصر باستخدام Arrays.copyOf().

1. تنفيذ برنامج يمكنه إدخال تعبير في العلامة اللاحقة وإخراج قيمته:

import java.util.Scanner;

public class PostfixEvaluator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);