## Topics

1)interface Stack<E> {

boolean isEmpty();

int size();

E top();

void push(E e);

E pop();

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

* isEmpty()
* size()

2) class ArrayStack<E> implements Stack<E> {

private E[] data;

private int top;

@SuppressWarnings("unchecked")

public ArrayStack(int capacity) {

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

top = -1;

}

public boolean isEmpty() {

return top == -1;

}

public int size() {

return top + 1;

}

public E top() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty");

}

return data[top];

}

public void push(E e) {

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

throw new IllegalStateException("Stack is full");

}

data[++top] = e;

}

public E pop() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty");

}

E popped = data[top];

data[top--] = null;

return popped;

}

}

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

## 

class LinkedStack<E> implements Stack<E> {

private static class Node<E> {

E value;

Node<E> next;

Node(E value) {

this.value = value;

}

}

private Node<E> top;

private int size;

public boolean isEmpty() {

return size == 0;

}

public int size() {

return size;

}

public E top() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty");

}

return top.value;

}

public void push(E e) {

Node<E> newNode = new Node<>(e);

newNode.next = top;

top = newNode;

size++;

}

public E pop() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty");

}

E popped = top.value;

top = top.next;

size--;

return popped;

}

}

1. import java.util.Stack;

public class Transfer {

public static Stack<Integer> stackPush(Stack<Integer> stack) {

for (int i = 0; i < 5; i++) {

Integer push = stack.push(i);

System.out.println(push);

}

return stack;

}

public static Stack<Integer> transfer(Stack<Integer> stack1, Stack<Integer> stack2) {

stack2 = stackPush(stack1);

System.out.println("Stack transferred successfully: " + stack2);

return stack2;

}

public static void main(String[] args) {

Stack<Integer> stack1 = new Stack<>();

Stack<Integer> stack2 = new Stack<>();

Stack<Integer> finalStack = transfer(stack1, stack2);

}

}

## 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.

import java.util.Stack;

public class StackTransfer {

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

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

// Transfer all elements from S to a temporary stack

while (!S.isEmpty()) {

temp.push(S.pop());

}

// Transfer all elements from the temporary stack to T

while (!temp.isEmpty()) {

T.push(temp.pop());

}

}

public static void main(String[] args) {

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

S.push(1);

S.push(2);

S.push(3);

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

transfer(S, T);

System.out.println("Stack T elements:");

while (!T.isEmpty()) {

System.out.println(T.pop());

}

}

}

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

import java.util.Stack;

public class StackEmpty {

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

if (!stack.isEmpty()) {

stack.pop();

empty(stack);

}

}

public static void main(String[] args) {

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

stack.push(1);

stack.push(2);

stack.push(3);

empty(stack);

System.out.println("Is the stack empty? " + stack.isEmpty());

}

}

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.

import java.util.Stack;

public class PostfixEvaluation {

public static int evaluate(String expression) {

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

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

char c = expression.charAt(i);

if (Character.isDigit(c)) {

stack.push(c - '0');

} else {

int val1 = stack.pop();

int val2 = stack.pop();

switch (c) {

case '+':

stack.push(val2 + val1);

break;

case '-':

stack.push(val2 - val1);

break;

case '/':

stack.push(val2 / val1);

break;

case '\*':

stack.push(val2 \* val1);

break;

}

}

}

return stack.pop();

}

public static void main(String[] args) {

String expression = "52+83-\*4/";

System.out.println("The result of the postfix expression " + expression + " is: " + evaluate(expression));

}

}

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

import java.util.Stack;

public class ArrayStack<E> implements Cloneable {

private Stack<E> stack;

public ArrayStack() {

this.stack = new Stack<>();

}

public void push(E e) {

stack.push(e);

}

public E pop() {

return stack.pop();

}

// دالة clone() الجديدة

@Override

public ArrayStack<E> clone() {

try {

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

cloneStack.stack = (Stack<E>) this.stack.clone();

return cloneStack;

} catch (CloneNotSupportedException e) {

throw new AssertionError();

}

}

}

5.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 PostfixEvaluation {

public static int evaluate(String expression) {

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

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

char c = expression.charAt(i);

if (Character.isDigit(c)) {

stack.push(c - '0');

} else {

int val1 = stack.pop();

int val2 = stack.pop();

switch (c) {

case '+':

stack.push(val2 + val1);

break;

case '-':

stack.push(val2 - val1);

break;

case '/':

stack.push(val2 / val1);

break;

case '\*':

stack.push(val2 \* val1);

break;

}

}

}

return stack.pop();

}

public static void main(String[] args) {

String expression = "52+83-\*4/";

System.out.println("The result of the postfix expression " + expression + " is: " + evaluate(expression));

}

}