## 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.

الحل

import java.util.Stack;

public class StackTransfer {

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

while (!source.isEmpty()) {

T element = source.pop();

destination.push(element);

}

}

}

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

الحل

import java.util.Stack;

public class StackRemoval {

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

if (stack.isEmpty()) {

return;

}

stack.pop();

removeAll(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.

الحل

import java.util.Stack;

public class PostfixEvaluation {

public static int evaluatePostfix(String postfixExpression) {

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

for (char c : postfixExpression.toCharArray()) {

if (Character.isDigit(c)) {

int operand = Character.getNumericValue(c);

stack.push(operand);

} else if (isOperator(c)) {

int operand2 = stack.pop();

int operand1 = stack.pop();

int result = performOperation(c, operand1, operand2);

stack.push(result);

}

}

return stack.pop();

}

private static boolean isOperator(char c) {

return c == '+' || c == '-' || c == '\*' || c == '/';

}

private static int performOperation(char 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("Invalid operator: " + operator);

}

}

}

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

الحل

import java.util.Arrays;

public class ArrayStack<T> implements Cloneable {

private Object[] elements;

private int size;

public ArrayStack() {

elements = new Object[10];

size = 0;

}

@Override

public ArrayStack<T> clone() {

try {

ArrayStack<T> clonedStack = (ArrayStack<T>) super.clone();

clonedStack.elements = Arrays.copyOf(elements, size);

return clonedStack;

} catch (CloneNotSupportedException e) {

throw new InternalError(e);

}

}

// الأساليب الأخرى لفئة ArrayStack مثل push() و pop() و isEmpty() وغيرها...

}

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

الحل

import java.util.Stack;

public class PostfixEvaluator {

public static int evaluatePostfix(String postfixExpression) {

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

for (char c : postfixExpression.toCharArray()) {

if (Character.isDigit(c)) {

int operand = Character.getNumericValue(c);

stack.push(operand);

} else if (isOperator(c)) {

int operand2 = stack.pop();

int operand1 = stack.pop();

int result = performOperation(c, operand1, operand2);

stack.push(result);

}

}

return stack.pop();

}

private static boolean isOperator(char c) {

return c == '+' || c == '-' || c == '\*' || c == '/';

}

private static int performOperation(char 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("Invalid operator: " + operator);

}

}

public static void main(String[] args) {

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

int result = evaluatePostfix(postfixExpression);

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

}

}