## Topics

1. Create Stack Interface

java

public interface Stack<E> {

E pop();

E peek();

int size();

boolean isEmpty();

1. Create Stack Using Array

public class Stack {

private int[] stackArray;

private int top;

private int capacity;

public Stack(int capacity) {

this.capacity = capacity;

this.stackArray = new int[capacity];

this.top = -1;

}

public boolean isEmpty() {

return top == -1;

}

public boolean isFull() {

return top == capacity - 1;

}

public void push(int item) {

if (isFull()) {

System.out.println("Stack is full. Cannot push item: " + item);

} else {

stackArray[++top] = item;

System.out.println("Pushed item: " + item);

}

}

public int pop() {

if (isEmpty()) {

System.out.println("Stack is empty. Cannot pop item.");

return -1;

} else {

int poppedItem = stackArray[top--];

System.out.println("Popped item: " + poppedItem);

return poppedItem;

}

}

public int peek() {

if (isEmpty()) {

System.out.println("Stack is empty. No item to peek.");

return -1;

} else {

return stackArray[top];

}

}

}

public static void main(String[] args) {

Stack stack = new Stack(5);

stack.push(1);

stack.push(2);

stack.push(3);

System.out.println("Peek: " + stack.peek());

System.out.println("Size: " + stack.size());

stack.pop();

stack.pop();

stack.pop();

System.out.println("Is Empty: " + stack.isEmpty());

}

}

1. Create Stack Using Linked Lists

public class Stack {

private Node top;

private class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

public boolean isEmpty() {

return top == null;

}

public void push(int item) {

Node newNode = new Node(item);

newNode.next = top;

top = newNode;

System.out.println("Pushed item: " + item);

}

public int pop() {

if (isEmpty()) {

System.out.println("Stack is empty. Cannot pop item.");

return -1;

} else {

int poppedItem = top.data;

top = top.next;

System.out.println("Popped item: " + poppedItem);

return poppedItem;

}

}

public int size() {

int count = 0;

Node current = top;

while (current != null) {

count++;

current = current.next;

}

return count;

}

public static void main(String[] args) {

Stack stack = new Stack();

stack.push(1);

stack.push(2);

stack.push(3);

System.out.println("Peek: " + stack.peek());

System.out.println("Size: " + stack.size());

stack.pop();

stack.pop();

stack.pop();

System.out.println("Is Empty: " + stack.isEmpty());

}

}

1. Implement Basic Methods of Stack

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

import java.util.ArrayList;

import java.util.List;

public class Stack<E> {

private List<E> stackList;

public Stack() {

stackList = new ArrayList<>();

}

public boolean isEmpty() {

return stackList.isEmpty();

}

public int size() {

return stackList.size();

}

public E top() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty.");

}

return stackList.get(stackList.size() - 1);

}

public void push(E item) {

stackList.add(item);

}

public E pop() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty.");

}

return stackList.remove(stackList.size() - 1);

}

public static void main(String[] args) {

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

System.out.println("Is Empty: " + stack.isEmpty());

System.out.println("Size: " + stack.size());

stack.push(1);

stack.push(2);

stack.push(3);

System.out.println("Top: " + stack.top());

System.out.println("Size: " + stack.size());

int poppedItem = stack.pop();

System.out.println("Popped item: " + poppedItem);

System.out.println("Top: " + stack.top());

System.out.println("Size: " + stack.size());

}

}

## 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 {

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

// Transfer elements from source stack to a temporary stack

Stack<E> tempStack = new Stack<>();

while (!source.isEmpty()) {

tempStack.push(source.pop());

}

// Transfer elements from the temporary stack to the destination stack

while (!tempStack.isEmpty()) {

destination.push(tempStack.pop());

}

}

public static void main(String[] args) {

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

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

// Push elements to the source stack

sourceStack.push(1);

sourceStack.push(2);

sourceStack.push(3);

System.out.println("Source Stack: " + sourceStack);

System.out.println("Destination Stack before transfer: " + destinationStack);

// Transfer elements from source stack to destination stack

transfer(sourceStack, destinationStack);

System.out.println("Destination Stack after transfer: " + destinationStack);

}

}

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

public class StackRemoval {

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

if (!stack.isEmpty()) {

stack.pop();

removeAllElements(stack);

}

}

public static void main(String[] args) {

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

stack.push(1);

stack.push(2);

stack.push(3);

System.out.println("Stack before removal: " + stack);

removeAllElements(stack);

System.out.println("Stack after removal: " + 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.

public class PostfixEvaluation {

public static int evaluate(String expression) {

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

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

char ch = expression.charAt(i);

if (Character.isDigit(ch)) {

// If the character is a digit, convert it to an integer and push it onto the stack

stack.push(ch - '0');

} else if (ch == ' ') {

// Ignore whitespace characters

continue;

} else {

// Operator encountered, pop the top two operands and perform the operation

int operand2 = stack.pop();

int operand1 = stack.pop();

int result;

switch (ch) {

case '+':

result = operand1 + operand2;

break;

case '-':

result = operand1 - operand2;

break;

case '\*':

result = operand1 \* operand2;

break;

case '/':

result = operand1 / operand2;

break;

default:

throw new IllegalArgumentException("Invalid operator: " + ch);

}

// Push the result back onto the stack

stack.push(result);

}

}

// The final result will be the only element left on the stack

return stack.pop();

}

public static void main(String[] args) {

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

int result = evaluate(postfixExpression);

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

}

}

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

import java.util.Arrays;

public class ArrayStack<E> implements Cloneable {

private Object[] elements;

private int size;

private static final int DEFAULT\_CAPACITY = 10;

public ArrayStack() {

elements = new Object[DEFAULT\_CAPACITY];

size = 0;

}

public void push(E element) {

ensureCapacity(size + 1);

elements[size] = element;

size++;

}

public E pop() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty.");

}

E element = peek();

elements[size - 1] = null;

size--;

return element;

}

public E peek() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty.");

}

return (E) elements[size - 1];

}

public boolean isEmpty() {

return size == 0;

}

public int size() {

return size;

}

private void ensureCapacity(int capacity) {

if (capacity > elements.length) {

int newCapacity = elements.length \* 2;

elements = Arrays.copyOf(elements, newCapacity);

}

}

@Override

public ArrayStack<E> clone() {

ArrayStack<E> clonedStack = new ArrayStack<>();

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

clonedStack.size = size;

return clonedStack;

}

public static void main(String[] args) {

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

stack.push(1);

stack.push(2);

stack.push(3);

ArrayStack<Integer> clonedStack = stack.clone();

System.out.println("Original Stack: " + stack.peek() + ", Size: " + stack.size());

System.out.println("Cloned Stack: " + clonedStack.peek() + ", Size: " + clonedStack.size());

}

}

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

public class PostfixExpressionEvaluator {

public static int evaluatePostfixExpression(String expression) {

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

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

char ch = expression.charAt(i);

if (Character.isDigit(ch)) {

// If the character is a digit, convert it to an integer and push it onto the stack

stack.push(ch - '0');

} else if (ch == ' ') {

// Ignore whitespace characters

continue;

} else {

// Operator encountered, pop the top two operands and perform the operation

int operand2 = stack.pop();

int operand1 = stack.pop();

int result;

switch (ch) {

case '+':

result = operand1 + operand2;

break;

case '-':

result = operand1 - operand2;

break;

case '\*':

result = operand1 \* operand2;

break;

case '/':

result = operand1 / operand2;

break;

default:

throw new IllegalArgumentException("Invalid operator: " + ch);

}

// Push the result back onto the stack

stack.push(result);

}

}

// The final result will be the only element left on the stack

return stack.pop();

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the postfix expression: ");

String postfixExpression = scanner.nextLine();

int result = evaluatePostfixExpression(postfixExpression);

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

}

}