STACKS

/\*

* @Duplicate Parentheses
* Givee a balanced expression, find if contains duplicate parentheses or not. A set of
* parentheses are duplicate if the same subexpression is surrounded by multiple parentheses.
* -> Return a true if it contains duplicates else return false.
* example:
* (i) -> (((a+(b)))+(c+d)) => true
* (ii) -> ((((a)+(b))+c+d)) => true
* (iii)-> ((a+b)+(c+d)) => false
* (iv) -> (((a+b))+c) => true

\*/

import java.util.Stack;;

public class DuplicateParenthesesCode {

public static boolean isDuplicate(String str) {

Stack<Character> s = new Stack<>();

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

char ch = str.charAt(i);

// closing

if (ch == ')') {

int count = 0;

while (s.peek() != '(') {

s.pop();

count++;

}

if (count < 1) {

return true; // duplicate

} else {

s.pop(); // opening pair

}

} else {

// opening

s.push(ch);

}

}

return false;

}

public static void main(String[] args) {

String str1 = "(((a+(b)))+(c+d))";

System.out.println(isDuplicate(str1)); // false

String str2 = "((a+b)+(c+d))";

System.out.println(isDuplicate(str2)); // true

}

}

2. /\*

* Push at the Bottom of the Stack

\*/

import java.util.Stack;

public class PushAtBottomOfStack {

public static void pushBottom(Stack<Integer> s, int data) {

if (s.isEmpty()) {

s.push(data);

return;

}

int pop = s.pop();

pushBottom(s, data);

s.push(pop);

}

public static void main(String[] args) {

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

s.push(1);

s.push(2);

s.push(3);

pushBottom(s, 4);

while (!s.isEmpty()) {

System.out.println(s.peek());

s.pop();

}

}

}

/\*

* Output:
* 3
* 2
* 1
* 4

\*/

3.

public class StackUsingLinkedList {

static class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

public static class Stack {

static Node head = null;

public boolean isEmpty() {

return head == null;

}

// push

public void push(int data) {

Node nextNode = new Node(data);

if (isEmpty()) {

head = nextNode;

return;

}

nextNode.next = head;

head = nextNode;

}

// pop

public int pop() {

if (isEmpty()) {

return -1;

}

int top = head.data;

head = head.next;

return top;

}

// peek

public int peek() {

if (isEmpty()) {

return -1;

}

return head.data;

}

}

public static void main(String[] args) {

Stack s = new Stack();

s.push(1);

s.push(2);

s.push(3);

s.push(4);

while (!s.isEmpty()) {

System.out.println(s.peek());

s.pop();

}

}

}

4.

/\*

* Create a Stack using two queues \*/

import java.util.LinkedList; import java.util.Queue;

public class StackUsingTwoQueues {

static class Stack {

Queue<Integer> q1 = new LinkedList<>();

Queue<Integer> q2 = new LinkedList<>();

public boolean isEmpty() {

return q1.isEmpty() && q2.isEmpty();

}

// push

public void push(int data) {

if (!q1.isEmpty()) {

q1.add(data);

} else {

q2.add(data);

}

}

int top = -1;

// pop

public int pop() {

if (q1.isEmpty() && q2.isEmpty()) {

System.out.println("Stack is Empty");

return -1;

}

if (!q1.isEmpty()) {

while (!q1.isEmpty()) {

top = q1.remove();

if (q1.isEmpty()) {

break;

}

q2.add(top);

}

} else {

while (!q2.isEmpty()) {

top = q2.remove();

if (q2.isEmpty()) {

break;

}

q1.add(top);

}

}

return top;

}

// peek

public int peek() {

if (q1.isEmpty() && q2.isEmpty()) {

System.out.println("Stack is Empty");

return -1;

}

int top = -1;

if (!q1.isEmpty()) {

while (!q1.isEmpty()) {

top = q1.remove();

q2.add(top);

}

} else {

while (!q2.isEmpty()) {

top = q2.remove();

q1.add(top);

}

}

return top;

}

}

public static void main(String[] args) {

Stack s = new Stack();

s.push(1);

s.push(2);

s.push(3);

s.push(4);

s.push(5);

while (!s.isEmpty()) {

System.out.println(s.peek());

s.pop();

}

}

}

/\*

* Output:
* 5
* 4
* 3
* 2
* 1 \*/