Assignment -3

**1. Implement a Stack using an array.**

* **Test Case 1**:  
  Input: Push 5, 3, 7, Pop  
  Output: Stack = [5, 3], Popped element = 7
* **Test Case 2**:  
  Input: Push 10, Push 20, Pop, Push 15  
  Output: Stack = [10, 15], Popped element = 20

class SimpleStack {

private int[] stack; // Array to hold stack elements

private int top; // Index of the top element

private int maxSize; // Maximum size of the stack

// Constructor to initialize the stack

public SimpleStack(int size) {

maxSize = size; // Set maximum size

stack = new int[maxSize]; // Initialize the stack array

top = -1; // Start with an empty stack

}

// Method to add an element to the stack

public void push(int value) {

if (top < maxSize - 1) { // Check if there is space

stack[++top] = value; // Increment top and add value

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

} else {

System.out.println("Stack is full! Cannot push " + value);

}

}

// Method to remove and return the top element

public int pop() {

if (top >= 0) { // Check if the stack is not empty

int poppedValue = stack[top--]; // Get the top value and decrement top

System.out.println("Popped: " + poppedValue);

return poppedValue; // Return the popped value

} else {

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

return -1; // Return -1 to indicate the stack is empty

}

}

// Method to display current stack elements

public void display() {

System.out.print("Stack = [");

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

System.out.print(stack[i]); // Print each element

if (i < top) {

System.out.print(", "); // Add a comma between elements

}

}

System.out.println("]"); // End of stack display

}

// Main method to test the stack functionality

public static void main(String[] args) {

SimpleStack stack = new SimpleStack(5); // Create a stack with a max size of 5

// Test Case 1

stack.push(5); // Push 5

stack.push(3); // Push 3

stack.push(7); // Push 7

stack.pop(); // Pop the top element

stack.display(); // Display current stack

// Test Case 2

stack.push(10); // Push 10

stack.push(20); // Push 20

stack.pop(); // Pop the top element

stack.push(15); // Push 15

stack.display(); // Display current stack

}}

**2 . Check for balanced parentheses using a stack.**

* **Test Case 1**:  
  Input: "({[()]})"  
  Output: Balanced
* **Test Case 2**:  
  Input: "([)]"  
  Output: Not Balanced

**ANS**:

import java.util.Stack;

public class BalancedParentheses {

// Method to check if the parentheses are balanced

public static String checkBalanced(String expression) {

Stack<Character> stack = new Stack<>(); // Create a stack to hold opening parentheses

// Loop through each character in the expression

for (char ch : expression.toCharArray()) {

// Push opening parentheses onto the stack

if (ch == '(' || ch == '{' || ch == '[') {

stack.push(ch);

}

// Check for closing parentheses

else if (ch == ')' || ch == '}' || ch == ']') {

// If the stack is empty or the top of the stack does not match, it's not balanced

if (stack.isEmpty()) {

return "Not Balanced";

}

char top = stack.pop(); // Pop the top of the stack

if (!isMatchingPair(top, ch)) {

return "Not Balanced";

}

}

}

// If the stack is empty, all parentheses are balanced

return stack.isEmpty() ? "Balanced" : "Not Balanced";

}

// Helper method to check if the opening and closing parentheses match

private static boolean isMatchingPair(char open, char close) {

return (open == '(' && close == ')') ||

(open == '{' && close == '}') ||

(open == '[' && close == ']');

}

// Main method to test the balanced parentheses function

public static void main(String[] args) {

// Test Case 1

String input1 = "({[()]})";

System.out.println("Input: " + input1);

System.out.println("Output: " + checkBalanced(input1)); // Expected output: Balanced

// Test Case 2

String input2 = "([)]";

System.out.println("Input: " + input2);

System.out.println("Output: " + checkBalanced(input2)); // Expected output: Not Balanced

}

}

3. Reverse a string using a stack.

• Test Case 1: Input: "hello" Output: "olleh"

• Test Case 2: Input: "world" Output: "dlrow"

ANS: import java.util.Stack;

public class ReverseStringUsingStack {

// Method to reverse a string using a stack

public static String reverseString(String input) {

Stack<Character> stack = new Stack<>(); // Create a stack to hold characters

// Push each character of the input string onto the stack

for (char ch : input.toCharArray()) {

stack.push(ch);

}

// StringBuilder to build the reversed string

StringBuilder reversed = new StringBuilder();

// Pop each character from the stack and append it to the StringBuilder

while (!stack.isEmpty()) {

reversed.append(stack.pop());

}

return reversed.toString(); // Convert StringBuilder to String

}

// Main method to test the reverse string function

public static void main(String[] args) {

// Test Case 1

String input1 = "hello";

System.out.println("Input: " + input1);

System.out.println("Output: " + reverseString(input1)); // Expected output: "olleh"

// Test Case 2

String input2 = "world";

System.out.println("Input: " + input2);

System.out.println("Output: " + reverseString(input2)); // Expected output: "dlrow"

}

}