Stacks

1. Decode Strings

```java
import java.util.Scanner;
import java.util.Stack;

public class DecodeString {
    public static String decode(String str) {
        Stack<Integer> integerStack = new Stack<>();
        Stack<String> stringStack = new Stack<>();
        String temp = "", result = "";

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

            // If the character is a digit
            if (Character.isDigit(str.charAt(i))) {
                while (Character.isDigit(str.charAt(i))) {
                    count = count * 10 + str.charAt(i) - '0'; // Use '0' to get the correct
integer

                    i++;
                }
                i--; // Decrease i to account for the last non-digit
                integerStack.push(count);
            }
            // If closing bracket ']' is found
            else if (str.charAt(i) == ']') {
                temp = "";
                count = integerStack.pop(); // Get the repeat count

                // Pop from stringStack until '['
                while (!stringStack.isEmpty() && !stringStack.peek().equals("[")) {
                    temp = stringStack.pop() + temp;
                }

                // Pop the opening bracket
                if (!stringStack.isEmpty() && stringStack.peek().equals("[")) {
                    stringStack.pop();
                }

                // Repeat temp count times and push back to stringStack
                StringBuilder repeatedString = new StringBuilder();
                for (int j = 0; j < count; j++) {
                    repeatedString.append(temp);
                }
                stringStack.push(repeatedString.toString());
            }
            // If opening bracket '[' is found
            else if (str.charAt(i) == '[') {
                stringStack.push("[");
            }
            // If it's a regular character
            else {
                stringStack.push(String.valueOf(str.charAt(i)));
            }
        }
```

```
        // Collect the final decoded string
        while (!stringStack.isEmpty()) {
            result = stringStack.pop() + result;
        }

        return result;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a String: ");
        String str = sc.next();
        System.out.println("After Decoding: " + decode(str));
    }
}
```

2. Duplicate Parenthesis

```
import java.util.Scanner;
import java.util.Stack;

public class DuplicateParenthesis {
    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 {
                s.push(ch); // Opening
            }
        }
        return false;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        System.out.print("Enter Strings: ");
        String str = sc.next();
        System.out.println("String is Duplicate. " + isDuplicate(str));
    }
}
```

3. Maximum Area in Histogram

```java
import java.util.Stack;

public class MaxAreaInHistogram {
    public static void maxArea (int arr[]) {
        int maxArea = 0;
        int nsr[] = new int[arr.length];
        int nsl[] = new int[arr.length];
        // Next Smaller Right
        Stack <Integer> s = new Stack<>();
        for (int i = arr.length-1; i >= 0; i--) {
            while (!s.isEmpty() && arr[s.peek()] >= arr[i]) {
                s.pop();
            }
            if (s.isEmpty()) {
                nsr[i] = arr.length;
            } else {
                nsr[i] = s.peek();
            }
            s.push(i);
        }
        // Next Smaller Left
        s = new Stack<>();
        for (int i = 0; i < arr.length; i++) {
            while (!s.isEmpty() && arr[s.peek()] >= arr[i]) {
                s.pop();
            }
            if (s.isEmpty()) {
                nsl[i] = -1;
            } else {
                nsl[i] = s.peek();
            }
            s.push(i);
        }
        for (int i = 0; i < arr.length; i++) {
            int height = arr[i];
            int width = nsr[i] - nsl[i] - 1;
            int currArea = height * width;
            maxArea = Math.max(currArea, maxArea);
        }
        System.out.println("Max Area in Histogram = " + maxArea);
    }
    public static void main(String[] args) {
        int arr[] = {2, 1, 5, 6, 2, 3};
        maxArea(arr);
    }
}
```

4. Next Greater Element

```java
import java.util.Stack;

public class NextGreaterElement {
    public static void main(String[] args) {
        int arr[] = {6, 8, 0, 1, 3};
        Stack <Integer> s = new Stack<>();
```

```
        int nxtGreater[] = new int [arr.length];
        for (int i = arr.length-1; i >= 0; i--) {
            while (!s.isEmpty() && arr[s.peek()] <= arr[i]) {
                s.pop();
            }
            if (s.isEmpty()) {
                nxtGreater[i] = -1;
            } else {
                nxtGreater[i] = arr[s.peek()];
            }
            s.push(i);
        }
        System.out.print("Next Greater Elements of Array is: ");
        for (int i = 0; i < nxtGreater.length; i++) {
            System.out.print(nxtGreater[i] + " ");
        }
        System.out.println();
    }
}
```

5. Palindrome Linked List using Stack

```
import java.util.Stack;

public class PalindromeLL {
    public static class Node {      // TC and SC = O(n)
        int data;
        Node ptr;
        Node (int d) {
            ptr = null;
            data = d;
        }
    }
    public static boolean isPalindrome (Node head) {
        Node slow = head;
        boolean ispalin = true;
        Stack <Integer> s = new Stack<Integer>();
        while (slow != null) {
            s.push(slow.data);
            slow = slow.ptr;
        }
        while (head != null) {
            int i = s.pop();
            if (head.data == i) {
                ispalin = true;
            } else {
                ispalin = false;
                break;
            }
            head = head.ptr;
        }
        return ispalin;
    }
    public static void main(String[] args) {
        Node one = new Node(1);
        Node two = new Node(2);
```

```
        Node three = new Node(3);
        Node four = new Node(4);
        Node five = new Node(3);
        Node six = new Node(2);
        Node seven = new Node(1);
        one.ptr = two;
        two.ptr = three;
        three.ptr = four;
        four.ptr = five;
        five.ptr = six;
        six.ptr = seven;
        boolean condition = isPalindrome(one);
        System.out.println("Palindrome: " + condition);
    }
}
```

6. Push At Bottom

```
import java.util.Stack;

public class PushAtBottom {
    public static void pushBottom (Stack <Integer> s, int data) {
        if (s.isEmpty()) {
            s.push(data);
            return;
        }
        int top = s.pop();
        pushBottom(s, data);
        s.push(top);
    }
    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.pop());
        }
    }
}
```

7. Reverse Stack

```
import java.util.Stack;

public class ReverseStack {
    public static void pushAtbottom(Stack<Integer> s, int data) {
        if (s.isEmpty()) {
            s.push(data);
            return;
        }
        int top = s.pop();
        pushAtbottom(s, data);
        s.push(top);
    }
```

```
    public static void reverseStack(Stack<Integer> s) {
        if (s.isEmpty()) {
            return;
        }
        int top = s.pop();
        reverseStack(s);
        pushAtbottom(s, top);
    }

    public static void printStack(Stack<Integer> s) {
        while (!s.isEmpty()) {
            System.out.println(s.pop());
        }
    }

    public static void main(String[] args) {
        Stack<Integer> s = new Stack<>();
        s.push(1);
        s.push(2);
        s.push(3);
        System.out.println("Original Stack: ");
        printStack(s);
        // Since the original stack is empty after printing, reinitialize it.
        s.push(1);
        s.push(2);
        s.push(3);
        reverseStack(s);
        System.out.println("Reversed Stack: ");
        printStack(s);
    }
}
```

8. Reverse String using Stack

```
import java.util.Scanner;
import java.util.Stack;

public class ReverseStringUsingStack {
    public static String reverseString (String str) {
        Stack <Character> s = new Stack<>();
        int idx = 0;
        while (idx < str.length()) {
            s.push(str.charAt(idx));
            idx++;
        }
        StringBuilder result = new StringBuilder(" ");
        while (!s.isEmpty()) {
            char curr = s.pop();
            result.append(curr);
        }
        return result.toString();
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        System.out.print("Enter String: ");
```

```
        String str = sc.next();
        String result = reverseString(str);
        System.out.print("Reversed String: " + result);
    }
}
```

9. Simplify Path

```
import java.util.Stack;

public class SimplifyPath {             // UNIX STYLE FOLDER NAME

    public static String simplify(String A) {
        Stack<String> st = new Stack<String>();
        String res = "/";
        int len_A = A.length();

        for (int i = 0; i < len_A; i++) {
            String dir = "";

            // Skip the '/' characters
            while (i < len_A && A.charAt(i) == '/') {
                i++;
            }

            // Read the current directory or file name
            while (i < len_A && A.charAt(i) != '/') {
                dir += A.charAt(i);
                i++;
            }

            // If the directory is "..", pop the stack (move up one directory)
            if (dir.equals("..")) {
                if (!st.empty()) {
                    st.pop();
                }
            }
            // If the directory is ".", do nothing (current directory)
            else if (dir.equals(".")) {
                continue;
            }
            // If the directory name is not empty, push it onto the stack
            else if (dir.length() != 0) {
                st.push(dir);
            }
        }

        // If the stack is empty, return "/"
        if (st.empty()) {
            return res;
        }

        // Build the final simplified path
        while (!st.empty()) {
            res = "/" + st.pop() + res;
```

```
        }

        // If the final path is empty, return "/"
        return res.equals("") ? "/" : res;
    }

    public static void main(String[] args) {
        String str = "/a/./b/../../c/";
        String res = simplify(str);
        System.out.println(res); // Output: "/c/"
    }
}
```

10. Stack using Array List

```
import java.util.ArrayList;

public class StackUsingAL {
    public static class Stack {
        public static ArrayList <Integer> list = new ArrayList<>();
        public static boolean isEmpty() {
            return list.size() == 0;
        }
        public static void push (int data) {
            list.add(data);
        }
        public static int pop() {
            if (isEmpty()) {
                return -1;
            }
            int top = list.get(list.size()-1);
            list.remove(list.size()-1);
            return top;
        }
        public static int peek() {
            if (isEmpty()) {
                return -1;
            }
            return list.get(list.size()-1);
        }
    }
    public static void main(String[] args) {
        Stack s = new Stack();
        s.push(1);
        s.push(2);
        s.push(3);
        while (!s.isEmpty()) {
            System.out.println(s.peek());
            s.pop();
        }
    }
}
```

11. Stack using JCF

```java
import java.util.Stack;

public class StackUsingJCF {
    public static void main(String[] args) {
        Stack <Integer> s = new Stack<>();
        s.push(1);
        s.push(2);
        s.push(3);
        while (!s.isEmpty()) {
            System.out.println(s.peek());
            s.pop();
        }
    }
}
```

12. Stack using Linked List

```java
import java.util.*;
public class StackUsingLL {
    public static class Node {
        int data;
        Node next;
        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
    public static Node head = null;
    public static boolean isEmpty() {
        return head == null;
    }
    public static void push (int data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
            head = newNode;
            return;
        }
        newNode.next = head;
        head = newNode;
    }
    public static int pop() {
        if (isEmpty()) {
            return -1;
        }
        int top = head.data;
        head = head.next;
        return top;
    }
    public static int peek() {
        if (isEmpty()) {
            return -1;
        }
        return head.data;
    }
```

```
    public static void main(String[] args) {
        StackUsingLL s = new StackUsingLL();
        s.push(1);
        s.push(2);
        s.push(3);
        while (!s.isEmpty()) {
            System.out.println(s.peek());
            s.pop();
        }
    }
}
```

13. Stock Span Problem

```
import java.util.Stack;

public class StockSpanProblem {
    public static void stockSpan (int stock[], int span[]) {
        Stack <Integer> s = new Stack<>();
        span[0] = 1;
        s.push(0);
        for (int i = 1; i < stock.length; i++) {
            int currPrice = stock[i];
            while (!s.isEmpty() && currPrice > stock[s.peek()]) {
                s.pop();
            }
            if (s.isEmpty()) {
                span[i] = i+1;
            } else {
                int prevHigh = s.peek();
                span[i] = i - prevHigh;
                s.push(i);
            }
        }
    }
    public static void main(String[] args) {
        int stocks[] = {100, 80, 60, 70, 60, 85, 100};
        int span[] = new int[stocks.length];
        stockSpan(stocks, span);
        for(int i = 0; i < span.length; i++) {
            System.out.print(span[i] + " ");
        }
    }
}
```

14. Trapping Rainwater

```
import java.util.Stack;

public class TrappingRainwater {
    public static int maxWater (int[] height) {
        Stack <Integer> stack = new Stack<>();
        int n = height.length;
        int ans = 0;
        for (int i = 0; i < n; i++) {
            while ((!stack.isEmpty()) && (height[stack.peek()] < height[i])) {
```

```
                int pop_height = height[stack.peek()];
                stack.pop();
                if (stack.isEmpty()) {
                    break;
                }
                int distance = i - stack.peek() - 1;
                int min_height = Math.min(height[stack.peek()], height[i]) - pop_height;
                ans += distance * min_height;
            }
            stack.push(i);
        }
        return ans;
    }
    public static void main(String[] args) {
        int arr[] = {7, 0, 4, 2, 5, 0, 6, 4, 0, 5};
        System.out.println(maxWater(arr));
    }
}
```

15. Valid Parenthesis

```
import java.util.Scanner;
import java.util.Stack;

public class ValidParenthesis {
    public static boolean isValid (String str) {
        Stack <Character> s = new Stack<>();
        for (int i = 0; i < str.length(); i++){
            char ch = str.charAt(i);
            if (ch == '(' || ch == '{' || ch == '[') {
                s.push(ch);
            } else {
                if (s.isEmpty()) {
                    return false;
                }
                if ((s.peek() == '(' && ch == ')') || (s.peek() == '{' && ch == '}') || (s.peek()
== '[' && ch == ']')) {
                    s.pop();
                } else {
                    return false;
                }
            }
        }
        if (s.isEmpty()) {
            return true;
        } else {
            return false;
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        System.out.print("Enter Parenthesis String: ");
        String str = sc.next();
        System.out.println("Parenthesis String is Valid. " + isValid(str));
    }
}
```