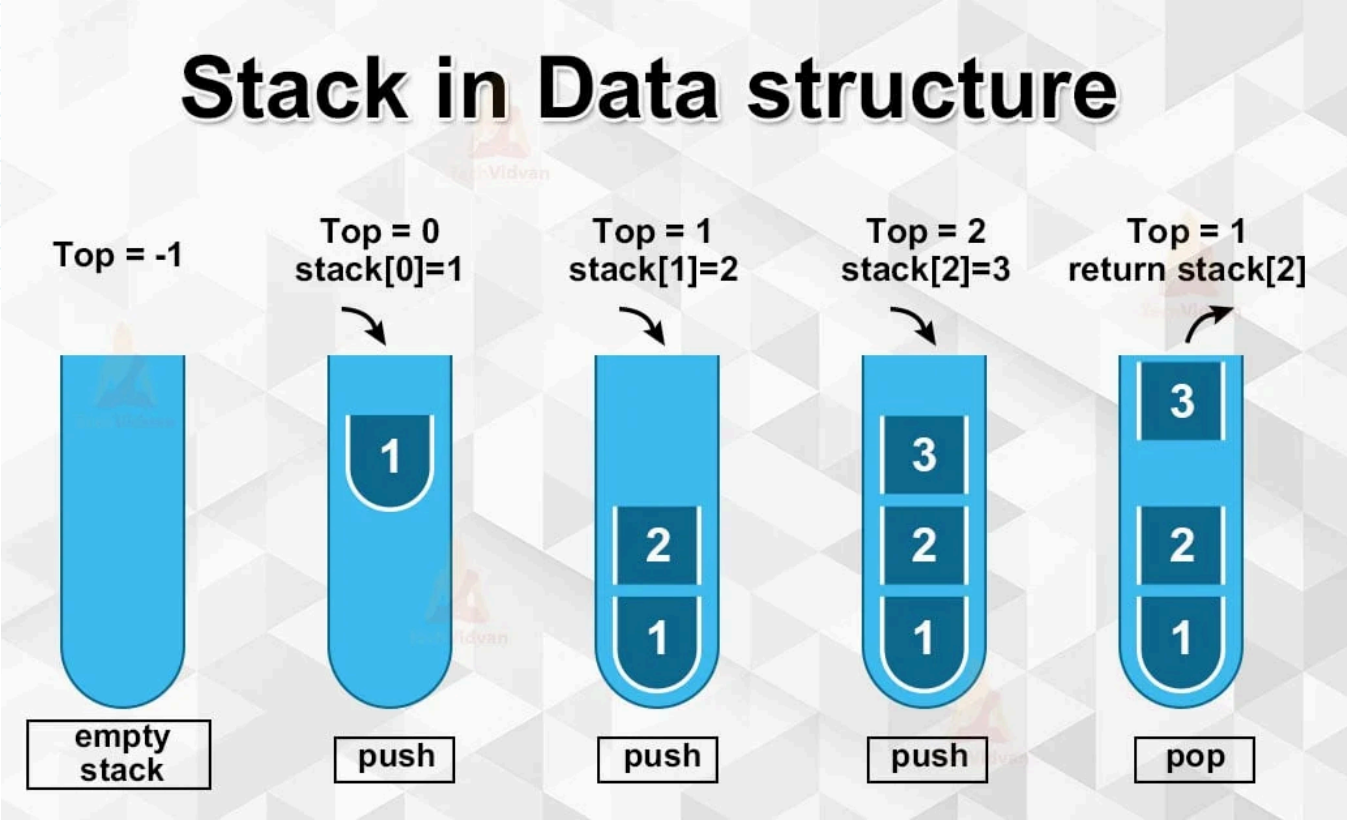
FAZAIA BILQUIS COLLEGE
NUR KHAN BASE,RWP



Task 4 – Stacks

SUBMITTED TO: *MA'AM SANA*

ROLL NO: *BSCS-13-F24-01*

SUBJECT: *DSA IN JAVA*

SUBMITTED BY: *MARIA ATTA*

LAB TASKS

TASK#1 STATIC ARRAY STACK (CAFETERIA TRAYS)

Scenario: In a cafeteria, trays are stacked at the counter.

- When a new clean tray comes → it's placed on top of the **stack (push)**.
- When a student takes one → it's taken from the **top (pop)**.
- If stack is full → **Overflow**
- If empty → **Underflow**

This models a **Last-In-First-Out (LIFO)** structure using a fixed-size array.

- **Program:**

TrayStackArray.java

```java
package Lab_04_Stacks;
import java.util.Scanner;
public class TrayStackArray {
    // Inner class to represent the Stack
    static class TrayStack {
        private final String[] data;  // array to store tray IDs
        private int top;              // index of top tray
        public TrayStack(int capacity) {
            data = new String[capacity];
            top = -1;
        }
        // Check if stack is empty
        public boolean isEmpty() {
            return top == -1;
        }
        // Check if stack is full
        public boolean isFull() {
            return top == data.length - 1;
        }
        // Push (Add tray)
        public void push(String value) {
            if (isFull()) {
                System.out.println("Overflow (stack full)");
                return;
            }
            data[++top] = value; // move top up and insert value
            System.out.println("Added tray " + value);
        }
        // Pop (Take tray)
        public void pop() {
            if (isEmpty()) {
                System.out.println("Underflow (no trays)");
                return;
            }
            System.out.println("Gave tray " + data[top--]);
        }
        // Peek (Show top tray without removing)
        public void peek() {
            if (isEmpty()) System.out.println("Stack empty");
            else System.out.println("Top tray: " + data[top]);
        }
        // Show all trays from top → bottom
        public void show() {
            if (isEmpty()) {
                System.out.println("[empty]");
                return;
            }
            System.out.print("Stack (top->bottom): ");
            for (int i = top; i >= 0; i--)
                System.out.print(data[i] + (i == 0 ? "" : " "));
            System.out.println();
        }
    }
    // Menu printer
    private static void printMenu() {
        System.out.println("\n== Cafeteria Tray Stack (Array) ==");
        System.out.println("1) Add tray");
        System.out.println("2) Take tray");
        System.out.println("3) Show top");
        System.out.println("4) Show all");
        System.out.println("0) Exit");
        System.out.print("Choose: ");
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter capacity K: ");
        int k = sc.nextInt();
        sc.nextLine(); // consume newline
        TrayStack stack = new TrayStack(k);
        printMenu();
        while (true) {
            String choice = sc.nextLine().trim();
            if (choice.isEmpty()) {
                System.out.print("Choose: ");
                continue;
            }
            char c = choice.charAt(0);
            if (c == '0') break;
            else if (c == '1') {
                System.out.print("Enter tray id (e.g., T1): ");
                String id = sc.next();
                sc.nextLine();
                stack.push(id);
            } else if (c == '2') {
                stack.pop();
            } else if (c == '3') {
                stack.peek();
            } else if (c == '4') {
                stack.show();
            } else {
                System.out.println("Invalid choice. Press 5 for help.");
            }
            System.out.print("Choose: ");
        }

        System.out.println("Goodbye!");
        sc.close();
    }
}
```

- **Output:**

```
Console
TrayStackArray [Java Application] C:\Users\manan\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.v
Enter capacity K: 3

== Cafeteria Tray Stack (Array) ==
1) Add tray
2) Take tray
3) Show top
4) Show all
0) Exit
Choose: 1
Enter tray id (e.g., T1): 32
Added tray 32
Choose: 1
Enter tray id (e.g., T1): 56
Added tray 56
Choose: 1
Enter tray id (e.g., T1): 45
Added tray 45
Choose: 1
Enter tray id (e.g., T1): 78
Overflow (stack full)
Choose: 2
Gave tray 45
Choose: 4
Stack (top->bottom): 56 32
Choose:
```

TASK 2: LINKED LIST STACK (BROWSER BACK HISTORY)

Scenario: When browsing the internet:

- Visiting a new page → **push page title.**
- Clicking "Back" → **pop from stack**.
- Show current page → **peek top of stack**.
- Show all visited pages → **display stack top→bottom**.

Here, the **stack is dynamic (no fixed size)**, implemented using **linked nodes**.

- **Program:**

BrowserHistoryLinkedStack.java

```java
package Lab_04_Stacks;
import java.util.Scanner;
public class BrowserHistoryLinkedStack {
    static class LinkedStack {
        private static class Node {
            String data; // page title
            Node next;   // pointer to next node
            Node(String data, Node next) {
                this.data = data;
                this.next = next;
            }
        }
        private Node head; // top of stack (most recent page)
        public boolean isEmpty() {
            return head == null;
        }
        public void push(String pageTitle) {
            head = new Node(pageTitle, head);
            System.out.println("Visited: " + pageTitle);
        }
        public void pop() {
            if (isEmpty()) {
                System.out.println("No history (at Home)");
                return;
            }
            String lastPage = head.data;
            head = head.next;
            System.out.println("Back to " + lastPage + ". (Now use CURRENT to see where you are)");
        }
        public void peek() {
            if (isEmpty()) System.out.println("Current page: Home");
            else System.out.println("Current page: " + head.data);
        }
        public void show() {
            if (isEmpty()) {
                System.out.println("[empty]");
                return;
            }
            StringBuilder sb = new StringBuilder();
```

```java
            Node cur = head;
            while (cur != null) {
                if (sb.length() > 0) sb.append(" ");
                sb.append(cur.data);
                cur = cur.next;
            }
            System.out.println("History (top->bottom): " + sb);
        }
    }
    private static void printMenu() {
        System.out.println("\n== Browser Back History (Linked List) ==");
        System.out.println("1) Visit page");
        System.out.println("2) Back");
        System.out.println("3) Current page");
        System.out.println("4) Show history");
        System.out.println("5) Help (show menu)");
        System.out.println("0) Exit");
        System.out.print("Choose: ");
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        LinkedStack history = new LinkedStack();
        System.out.println("Scenario: Unlimited page visits; use a linked-list stack for Back history.");
        printMenu();
        while (true) {
            String choice = sc.nextLine().trim();
            if (choice.isEmpty()) { System.out.print("Choose: "); continue; }
            char c = choice.charAt(0);
            if (c == '0') break;
            else if (c == '1') {
                System.out.print("Enter page title: ");
                String title = sc.nextLine().trim();
                if (title.isEmpty()) System.out.println("Please type a page title.");
                else history.push(title);
            } else if (c == '2') {
                history.pop();
            } else if (c == '3') {
                history.peek();
            } else if (c == '4') {
                history.show();
            } else if (c == '5') {
                printMenu();
            } else {
                System.out.println("Invalid choice. Press 5 for help.");
            }
            System.out.print("Choose: ");
        }
        System.out.println("Goodbye!");
        sc.close();
    }
}
```

- **Output:**

```
== Browser Back History (Linked List) ==
1) Visit page
2) Back
3) Current page
4) Show history
5) Help (show menu)
0) Exit
Choose: 1
Enter page title: 23
Visited: 23
Choose: 1
Enter page title: 45
Visited: 45
Choose: 1
Enter page title: 34
Visited: 34
Choose: 2
Back to 34. (Now use CURRENT to see where you are)
Choose: 3
Current page: 45
Choose: 4
History (top->bottom): 45 23
Choose: 5

== Browser Back History (Linked List) ==
1) Visit page
2) Back
3) Current page
4) Show history
5) Help (show menu)
0) Exit
Choose: Choose: 0
Goodbye!
```

TASK 3: STACK APPLICATION: INFIX → POSTFIX CONVERSION

Scenario: We often use stacks to convert infix expressions (like A + B * C) into postfix form (A B C * +).

- This helps compilers and calculators evaluate expressions correctly based on operator precedence.

- **Program:**

```java
InfixToPostfixMini.java
package Lab_04_Stacks;
import java.util.*;
public class InfixToPostfixMini {
    static int prec(String op) {
        switch (op) {
            case "^": return 3;
            case "*": case "/": case "%": return 2;
            case "+": case "-": return 1;
            default: return -1;
        }
    }
    static boolean isOp(String t) { // Check if token is operator
        return "+-*/%^".contains(t);
    }
    static boolean leftAssoc(String t) { // ^ is right associative; others are left
        return !t.equals("^");
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter infix (space-separated):");
        String[] tok = sc.nextLine().trim().split("\\s+");
        Stack<String> ops = new Stack<>();
        List<String> out = new ArrayList<>();
        try {
            for (String t : tok) {
                if (t.equals("(")) {
                    ops.push(t);
                } else if (t.equals(")")) {
                    while (!ops.isEmpty() && !ops.peek().equals("("))
                        out.add(ops.pop());
                    if (ops.isEmpty()) throw new RuntimeException("Mismatched ( )");
                    ops.pop(); // remove '('
                } else if (isOp(t)) {
                    while (!ops.isEmpty() && isOp(ops.peek())) {
                        String top = ops.peek();
                        int pt = prec(top), pc = prec(t);
                        if (pt > pc || (pt == pc && leftAssoc(t)))
                            out.add(ops.pop());
                        else break;
```

```java
                    }
                    ops.push(t);
                } else {
                    out.add(t);
                }
            }
            while (!ops.isEmpty()) {
                String op = ops.pop();
                if (op.equals("(")) throw new RuntimeException("Mismatched ( )");
                out.add(op);
            }
            System.out.println("Postfix: " + String.join(" ", out));

        } catch (RuntimeException e) {
            System.out.println("Error: " + e.getMessage());
        }
        sc.close();
    }
}
```

- **Output:**

```
Console
<terminated> InfixToPostfixMini [Java Application] C:\Users\manan\.p2\pool\plugins\org.eclipse.justj.openjd
Enter infix (space-separated):
a + b * 2
Postfix: a b 2 * +
```

HOME TASKS

1) ARRAY STACK – DECIMAL → BINARY → DECIMAL (ROUND-TRIP)

Scenario: You'll convert a number from **Decimal → Binary using a stack (LIFO order)**, then pop again to get back the original number.

- **Program:**

DecimalBinaryStack.java

```java
package Lab_04_Stacks;
import java.util.Scanner;
public class DecimalBinaryStack {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number: ");
        int n = sc.nextInt();
        if (n == 0) {
            System.out.println("Binary: 0");
            System.out.println("Reconstructed: 0");
            return;
        }
        int[] stack = new int[32];
        int top = -1, temp = n;
        while (temp > 0) { // Decimal → Binary
            if (top == 31) { System.out.println("Overflow"); return; }
            stack[++top] = temp % 2;
            temp /= 2;
        }
        System.out.print("Binary: ");
        for (int i = top; i >= 0; i--) System.out.print(stack[i]);
        System.out.println();
        int ans = 0; // Binary → Decimal
        for (int i = top; i >= 0; i--) ans = ans * 2 + stack[i];
        System.out.println("Reconstructed: " + ans);
    }
}
```

- **Output:**

Console

```
<terminated> DecimalBinaryStack [Java Application] C:\Users\manan\.p2\pool\plugins\org.eclipse.justj.open
Enter number: 45
Binary: 101101
Reconstructed: 45
```

2) ARRAY STACK – REVERSE WORDS

Scenario: Use stack to reverse order of words, not letters.

- **Program:**

ReverseWordsStack.java

```java
package Lab_04_Stacks;
import java.util.Scanner;
public class ReverseWordsStack {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] words = sc.nextLine().split(" ");
        String[] stack = new String[words.length];
        int top = -1;
        for (String w : words) stack[++top] = w;
        while (top >= 0) {
            System.out.print(stack[top--]);
            if (top >= 0) System.out.print(" ");
        }
    }
}
```

- **Output:**

```
Console
<terminated> ReverseWordsStack [Java Application] C:\Users\manan\.p2\pool\plugins\org.eclipse.just
this is easy
easy is this
```

3) ARRAY STACK – REDUNDANT PARENTHESES

Scenario: Parentheses are redundant if they enclose no operator.

- **Program:**

```java
RedundantParentheses.java
package Lab_04_Stacks;
import java.util.Scanner;
public class RedundantParentheses {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String exp = sc.nextLine();
        char[] stack = new char[100];
        int top = -1;
        for (char c : exp.toCharArray()) {
            if (c == ')') {
                boolean hasOp = false;
                while (top >= 0 && stack[top] != '(') {
                    char x = stack[top--];
                    if ("+-*/%^".indexOf(x) != -1) hasOp = true;
                }
                if (top < 0) { System.out.println("Invalid"); return; }
                top--; // pop '('
                if (!hasOp) { System.out.println("Redundant"); return; }
            } else {
                stack[++top] = c;
            }
        }
        System.out.println("OK");
    }
}
```

- **Output:**

```
Console
<terminated> RedundantParentheses [Java Application] C:\Users\manan\.p2\pool\plugins\org.eclipse.justj
(a + b ))
Invalid
```

4) ARRAY STACK – NEXT GREATER ELEMENT (NGE)

Scenario: Find next greater number for each element using a stack of indices.

- **Program:**

```java
NextGreaterElement.java
package Lab_04_Stacks;
import java.util.*;
public class NextGreaterElement {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) arr[i] = sc.nextInt();
        int[] ans = new int[n];
        Stack<Integer> s = new Stack<>();
        for (int i = 0; i < n; i++) {
            while (!s.isEmpty() && arr[i] > arr[s.peek()])
                ans[s.pop()] = arr[i];
            s.push(i);
        }
        while (!s.isEmpty()) ans[s.pop()] = -1;
        for (int i = 0; i < n; i++)
            System.out.println(arr[i] + " -> " + ans[i]);
    }
}
```

- **Output:**

```
Console
<terminated> NextGreaterElement [Java Application] C:\Users\manan\.p2\pool\plugins\org.eclipse.ju
6
2 5 7 8 10 45
2 -> 5
5 -> 7
7 -> 8
8 -> 10
10 -> 45
45 -> -1
```

5) LINKED-LIST STACK – O(1) MINIMUM

Scenario: Each node stores:

- **value**
- **minSoFar (minimum up to that node)**

So whenever we push, we calculate the min till now.

- **Program:**

```java
LinkedListStackMin.java
package Lab_04_Stacks;
import java.util.Scanner;
class Node {
    int value, minSoFar;
    Node next;
    Node(int v, int m, Node n) { value = v; minSoFar = m; next = n; }
}
public class LinkedListStackMin {
    static Node top = null;
    public static void push(int x) {
        int min = (top == null) ? x : Math.min(x, top.minSoFar);
        top = new Node(x, min, top);
    }
    public static void pop() {
        if (top == null) System.out.println("Underflow");
        else {
            System.out.println("POP -> " + top.value);
            top = top.next;
        }
    }
    public static void top() {
        if (top == null) System.out.println("Underflow");
        else System.out.println("TOP -> " + top.value);
    }
    public static void min() {
        if (top == null) System.out.println("Underflow");
        else System.out.println("MIN -> " + top.minSoFar);
    }
    public static void show() {
        Node temp = top;
        System.out.print("Stack: ");
        while (temp != null) {
            System.out.print(temp.value + " ");
            temp = temp.next;
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (true) {
            String cmd = sc.next();
            if (cmd.equalsIgnoreCase("PUSH")) push(sc.nextInt());
            else if (cmd.equalsIgnoreCase("POP")) pop();
            else if (cmd.equalsIgnoreCase("TOP")) top();
            else if (cmd.equalsIgnoreCase("MIN")) min();
            else if (cmd.equalsIgnoreCase("SHOW")) show();
            else break;
        }
    }
}
```

- **Output:**

```
Console
LinkedListStackMin [Java Application] C:\Users\manan\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.
PUSH 34
PUSH 56
TOP
TOP -> 56

MIN
MIN -> 34

SHOW
Stack: 56 34
```

6) ARRAY STACK – BACKSPACE TYPING SIMULATOR

Scenario: Use # to delete last character (like real typing).

- **Program:**

BackspaceSimulator.java

```java
package Lab_04_Stacks;
import java.util.Scanner;
public class BackspaceSimulator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
        char[] stack = new char[s.length()];
        int top = -1;
        for (char c : s.toCharArray()) {
            if (c == '#') {
                if (top >= 0) top--; // backspace
            } else {
                stack[++top] = c;
            }
        }
        for (int i = 0; i <= top; i++)
            System.out.print(stack[i]);
    }
}
```

- **OUTPUT:**

```
Console
<terminated> BackspaceSimulator [Java Application] C:\Users\manan\.p2\pool\plugins\org.eclipse.justj.open
ab#@c
a@c
```

7) ARRAY STACK – PATH SIMPLIFIER (UNIX-LIKE)

Scenario: Simplify paths like /a/./b/../../c/ to /c

- **Program:**

PathSimplifier.java

```java
package Lab_04_Stacks;
import java.util.*;
public class PathSimplifier {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String path = sc.nextLine();
        String[] parts = path.split("/");
        Stack<String> st = new Stack<>();
        for (String p : parts) {
            if (p.equals("") || p.equals(".")) continue;
            else if (p.equals("..")) {
                if (!st.isEmpty()) st.pop();
            } else st.push(p);
        }
        if (st.isEmpty()) System.out.println("/");
        else {
            StringBuilder sb = new StringBuilder();
            for (String dir : st) sb.append("/").append(dir);
            System.out.println(sb);
        }
    }
}
```

- **OUTPUT:**

```
Console
<terminated> PathSimplifier [Java Application] C:\Users\manan\.p2\pool\plugins\org.eclipse.justj.op
/a/./b/../../c/
/c
```

8) EXPRESSION CONVERSION – POSTFIX → INFIX

Scenario: When operator found → pop 2, make (a op b)

- **Program:**

PostfixToInfix.java

```java
package Lab_04_Stacks;
import java.util.*;
public class PostfixToInfix {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] tokens = sc.nextLine().split(" ");
        Stack<String> st = new Stack<>();
        for (String t : tokens) {
            if ("+-*/%^".contains(t)) {
                if (st.size() < 2) { System.out.println("Invalid"); re
                String b = st.pop(), a = st.pop();
                st.push("(" + a + " " + t + " " + b + ")");
            } else st.push(t);
        }
        if (st.size() != 1) System.out.println("Invalid");
        else System.out.println(st.pop());
    }
}
```

- **Output:**

```
Console
terminated> PostfixToInfix [Java Application] C:\Users\manan\.p2\pool\plugins\org.eclipse.justj.ope
A B + C *
(A + B) * C)
```

10) EXPRESSION CONVERSION – INFIX → PREFIX

Scenario: Trick: reverse → convert to postfix → reverse again.

- **Program:**

InfixToPrefix.java

```java
package Lab_04_Stacks;
import java.util.*;
public class InfixToPrefix {
    static int prec(String op) {
        if (op.equals("*") || op.equals("/")) return 2;
        if (op.equals("+") || op.equals("-")) return 1;
        return 0;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] tokens = sc.nextLine().split(" ");
        Collections.reverse(Arrays.asList(tokens));
        Stack<String> ops = new Stack<>();
        ArrayList<String> out = new ArrayList<>();
        for (int i = 0; i < tokens.length; i++) {
            String t = tokens[i];
            if (t.equals("(")) t = ")";
            else if (t.equals(")")) t = "(";
            if (t.matches("[A-Za-z0-9]+")) out.add(t);
            else if (t.equals("(")) ops.push(t);
            else if (t.equals(")")) {
                while (!ops.isEmpty() && !ops.peek().equals("("))
                    out.add(ops.pop());
                if (!ops.isEmpty()) ops.pop();
            } else { // operator
                while (!ops.isEmpty() && prec(ops.peek()) >= prec(t))
                    out.add(ops.pop());
                ops.push(t);
            }
        }
        while (!ops.isEmpty()) out.add(ops.pop());
        Collections.reverse(out);
        System.out.println(String.join(" ", out));
    }
}
```

- **OUTPUT:**

```
Console
<terminated> InfixToPrefix [Java Application] C:\Users\manan\.p2\pool\plugins\org.eclipse.justj.op
( A + B ) * ( C - D )
* + A B - C D
```

11) ARRAY STACK – BALANCED BRACKETS

Scenario: Push when open, pop when matching close.

- **Program:**

BalancedBrackets.java

```java
package Lab_04_Stacks;
import java.util.Scanner;
public class BalancedBrackets {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.nextLine();
        char[] stack = new char[s.length()];
        int top = -1;
        for (char c : s.toCharArray()) {
            if (c == '(' || c == '[' || c == '{') stack[++top] = c;
            else if (c == ')' || c == ']' || c == '}') {
                if (top < 0) { System.out.println("Not balanced");
                return;
                }
                char open = stack[top--];
                if ((c == ')' && open != '(') ||
                    (c == ']' && open != '[') ||
                    (c == '}' && open != '{')) {
                    System.out.println("Not balanced"); return;
                }
            }
        }
        System.out.println(top == -1 ? "Balanced" : "Not balanced");
    }
}
```

- **Output:**

```
Console
<terminated> BalancedBrackets [Java Application] C:\Users\manan\.p2\pool\plugins\org.ecli
( [ { } ] )
Balanced
```

12) ARRAY STACK – EVALUATE POSTFIX (INTEGERS)

Scenario: For numbers push, for operators pop 2 and evaluate.

- **Program:**

EvaluatePostfix.java

```java
package Lab_04_Stacks;
import java.util.*;
public class EvaluatePostfix {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] tokens = sc.nextLine().split(" ");
        Stack<Integer> st = new Stack<>();
        for (String t : tokens) {
            if (t.matches("-?\\d+")) st.push(Integer.parseInt(t));
            else if ("+-*/".contains(t)) {
                if (st.size() < 2) { System.out.println("Invalid");
                return;
                }
                int b = st.pop(), a = st.pop(), res = 0;
                switch (t) {
                    case "+": res = a + b; break;
                    case "-": res = a - b; break;
                    case "*": res = a * b; break;
                    case "/": res = a / b; break;
                }
                st.push(res);
            } else { System.out.println("Invalid token: " + t);
            return;
            }
        }
        if (st.size() == 1) System.out.println("Value: " + st.pop());
        else System.out.println("Invalid expression");
    }
}
```

- **OUTPUT:**

```
Console
<terminated> EvaluatePostfix [Java Application] C:\Users\manan\.p2\pool\
2 3 1 * + 9 -
Value: -4
```