Lab MST Assignment(Complex)

Student Name: Prateek Pratap Singh **UID: 22BCS10036**
Branch: BE-CSE **Section/Group: IOT_631-A**
Semester:6th **Date of Performance: 28/03/2025**
Subject Name: Project Based Learning in Java with Lab **Subject Code: 22CSH-359**

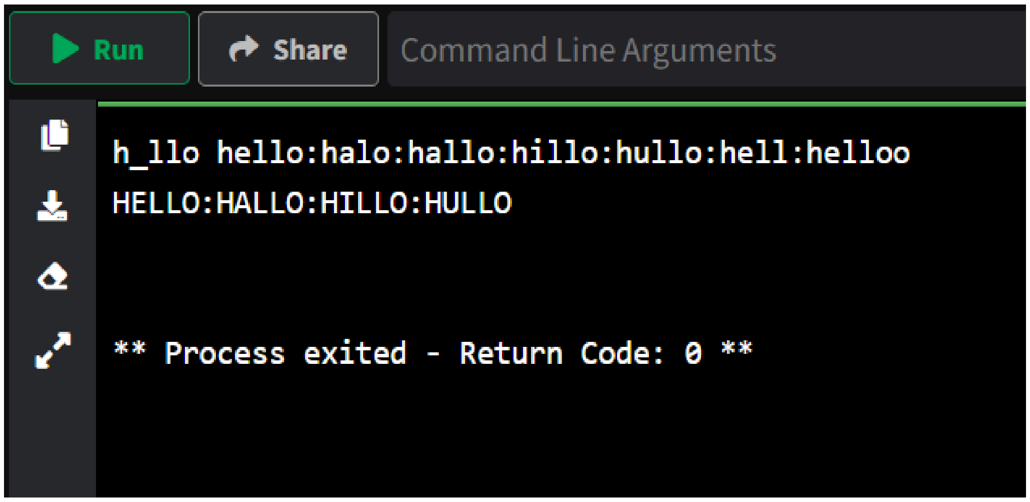
1. **(a) Aim:** Consider a function **public String matchFound(String input 1, String input 2),** where **input1** will contain only a single word with only 1 character replaces by an underscore '_' **input2** will contain a series of words separated by colons and no space character in between **input2** will not contain any other special character other than underscore and alphabetic characters. The methods should return output in a String type variable **"output1"** which contains all the words from input2 separated by colon which matches with input 1. All words in output1 should be in uppercase.

2. **Implementation:**

```
import java.util.*;

public class Main {
    public static String matchFound(String input1, String input2) {
        String[] words = input2.split(":");
        String pattern = input1.replace("_", ".");
        List<String> matchedWords = new ArrayList<>();
        for (String word : words) {
            if (word.matches(pattern)) {
                matchedWords.add(word.toUpperCase());
            }
        }
        return String.join(":", matchedWords);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String input1 = sc.next();
        String input2 = sc.next();
        System.out.println(matchFound(input1, input2));
        sc.close();
    }
}
```

3. **Output:**



```
h_llo hello:halo:hallo:hillo:hullo:hell:helloo
HELLO:HALLO:HILLO:HULLO


** Process exited - Return Code: 0 **
```

1. (b) Aim: String t is generated by random shuffling string s and then add one more letter at a random position.
Return the letter that was added to t.
Hint:
Input: s = "abcd", t = "abcde"
Output: "e"

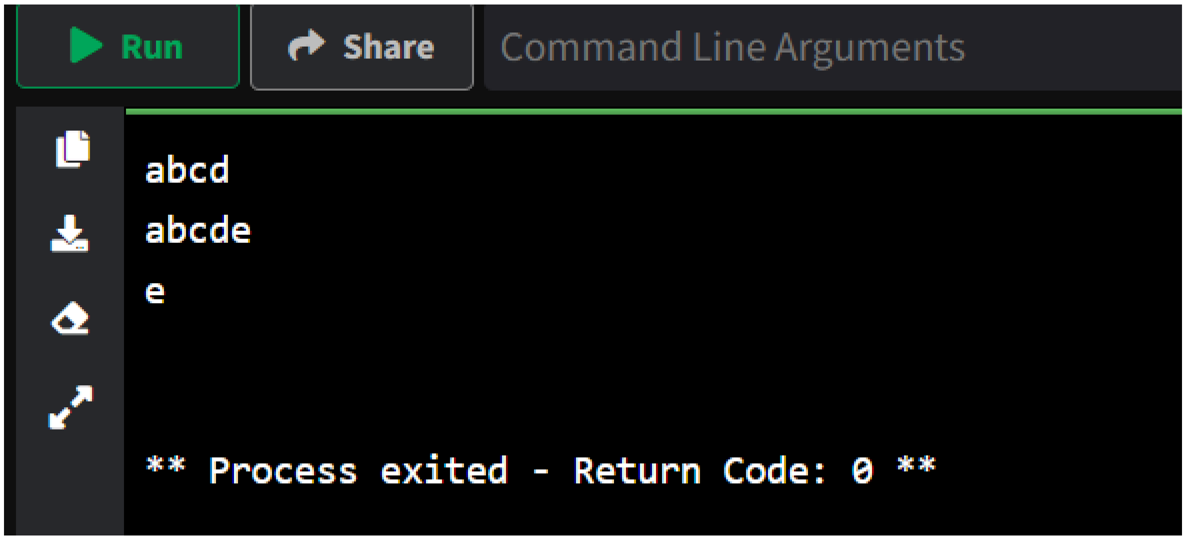
2. Implementation :

```
import java.util.*;

public class Main {
    public static char findTheDifference(String s, String t) {
        int sumS = 0, sumT = 0;
        for (char c : s.toCharArray()) sumS += c;
        for (char c : t.toCharArray()) sumT += c;
        return (char) (sumT - sumS);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        String t = sc.next();
        System.out.println(findTheDifference(s, t));
        sc.close();
    }
}
```

3. **Output :**



```
abcd
abcde
e


** Process exited - Return Code: 0 **
```

1. **Aim (c) :** The next greater element of some element x in an array is the first greater element that is to the right of x in the same array. You are given two distinct 0-indexed integer arrays nums1 and nums2, where nums1 is a subset of nums2. For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the next greater element of nums2[j] in nums2. If there is no next greater element, then the answer for this query is -1. Return an array ans of length nums1.length such that ans[i] is the next greater element as described above.
   Hint:
   Input: nums1 = [4,1,2], nums2 = [1,3,4,2]
   Output: [-1,3,-1]
   Explanation: The next greater element for each value of nums1 is as follows: - 4 is underlined in nums2 = [1,3,<u>4</u>,2]. There is no next greater element, so the answer is -1. - 1 is underlined in nums2 = [<u>1</u>,3,4,2]. The next greater element is 3. - 2 is underlined in nums2 = [1,3,4,<u>2</u>]. There is no next greater element, so the answer is -1.

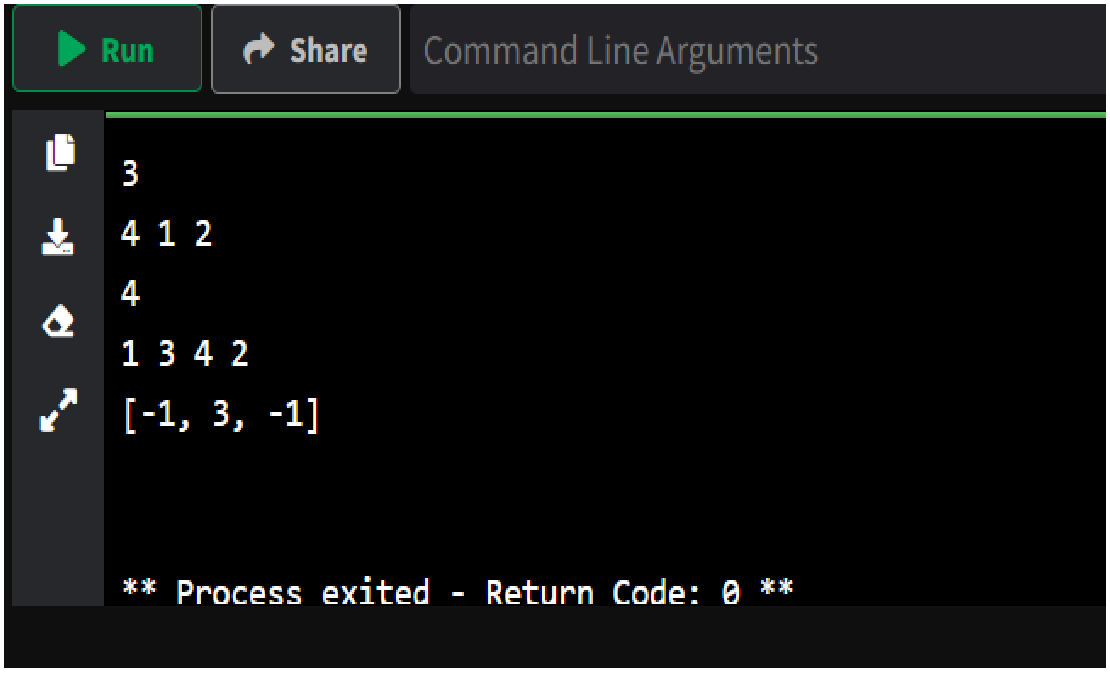
2. **Implementation :**

```
import java.util.*;

public class Main {
    public static int[] nextGreaterElement(int[] nums1, int[] nums2) {
        Map<Integer, Integer> nextGreaterMap = new HashMap<>();
        Stack<Integer> stack = new Stack<>();
        for (int num : nums2) {
            while (!stack.isEmpty() && stack.peek() < num) {
                nextGreaterMap.put(stack.pop(), num);
            }
            stack.push(num);
        }
        int[] result = new int[nums1.length];
        for (int i = 0; i < nums1.length; i++) {
            result[i] = nextGreaterMap.getOrDefault(nums1[i], -1);
        }
        return result;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n1 = sc.nextInt();
        int[] nums1 = new int[n1];
        for (int i = 0; i < n1; i++) nums1[i] = sc.nextInt();
        int n2 = sc.nextInt();
        int[] nums2 = new int[n2];
        for (int i = 0; i < n2; i++) nums2[i] = sc.nextInt();
        int[] result = nextGreaterElement(nums1, nums2);
        System.out.println(Arrays.toString(result));
        sc.close();
    }
}
```

3. Output:



```
3
4 1 2
4
1 3 4 2
[-1, 3, -1]


** Process exited - Return Code: 0 **
```

1. **Aim (d) :** A string containing only parentheses is balanced if the following is true: 1. if it is an empty string 2. if A and B are correct, AB is correct, 3. if A is correct, (A) and {A} and [A] are also correct. Examples of some correctly balanced strings are: "{}()", "[{()}]", "({()})" Examples of some unbalanced strings are: "{}(", "({)}", "[[", "}{" etc. Given a string, determine if it is balanced or not.
   Input Format :
   There will be multiple lines in the input file, each having a single non-empty string. You should read input till end-of-file.
   Output Format:
   For each case, print 'true' if the string is balanced, 'false' otherwise.
   Sample Input:
   {}() ({()}) {}([]
   Sample Output:
   true true false true

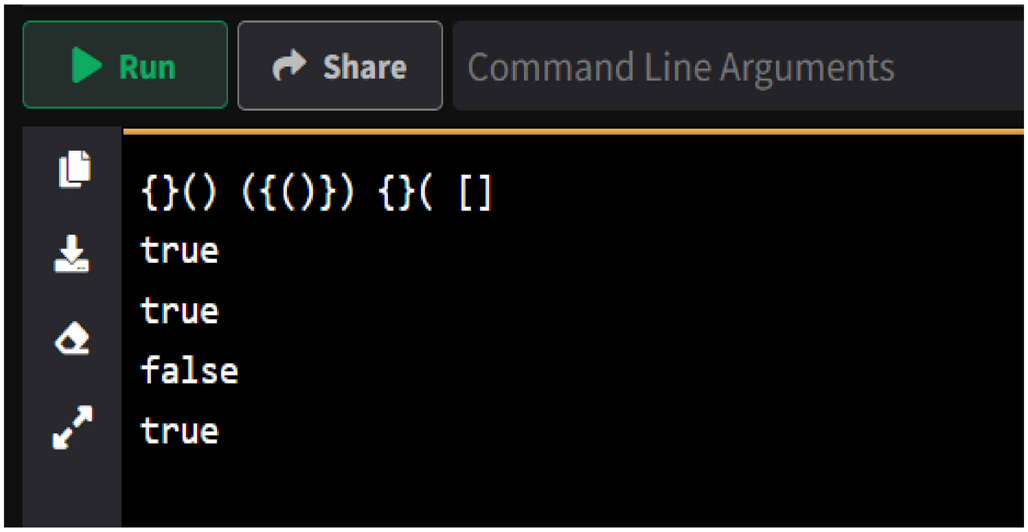
2. **Implementation :**

```
import java.util.*;

public class Main {
    public static boolean isBalanced(String s) {
        Stack<Character> stack = new Stack<>();
        for (char c : s.toCharArray()) {
            if (c == '(' || c == '{' || c == '[') {
                stack.push(c);
            } else if (c == ')' || c == '}' || c == ']') {
                if (stack.isEmpty()) return false;
                char top = stack.pop();
                if ((c == ')' && top != '(') || (c == '}' && top != '{') || (c == ']'
&& top != '[')) {
                    return false;
                }
            }
        }
        return stack.isEmpty();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            String s = sc.next();
            System.out.println(isBalanced(s));
        }
        sc.close();
    }
}
```

3. Output :



```
{}() ({()}) {}( []
true
true
false
true
```

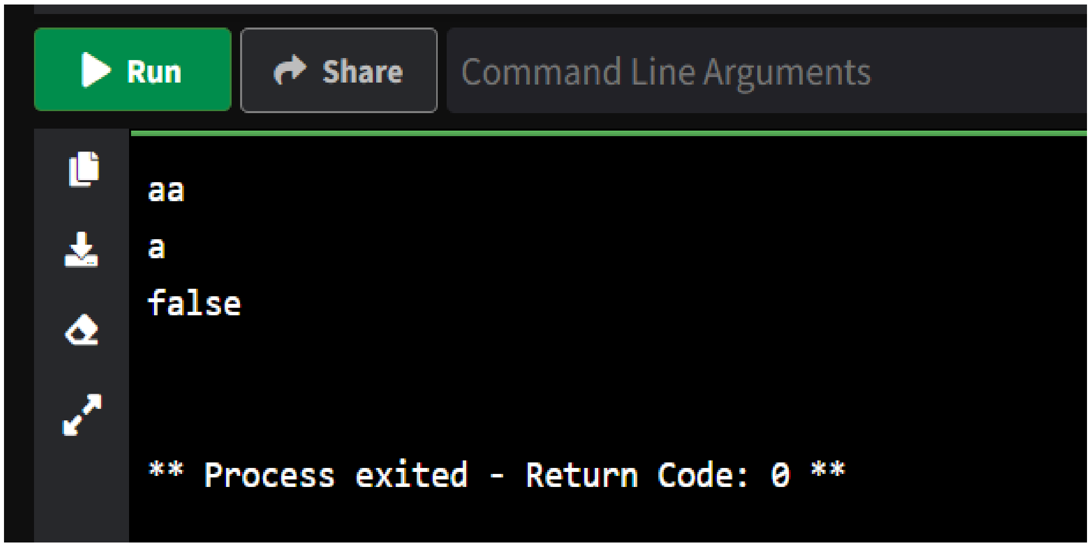
1. **Aim (e):** Given an input string (s) and a pattern (p), implement wildcard pattern matching with support for '?' and '*' where: '?' Matches any single character.
   '*' Matches any sequence of characters (including the empty sequence).
   The matching should cover the entire input string (not partial).
   Example 1:
   Input: s = "aa", p = "a"
   Output: false
   Explanation: "a" does not match the entire string "aa".
   Constraints:
   0 <= s.length, p.length <= 2000
   s contains only lowercase English letters.
   p contains only lowercase English letters, '?' or '*'.

2. **Implementation :**

```
import java.util.*;

public class Main {
    public static boolean isMatch(String s, String p) {
        int m = s.length(), n = p.length();
        boolean[][] dp = new boolean[m + 1][n + 1];
        dp[0][0] = true;
        for (int j = 1; j <= n; j++) {
            if (p.charAt(j - 1) == '*') dp[0][j] = dp[0][j - 1];
        }
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (p.charAt(j - 1) == '?' || p.charAt(j - 1) == s.charAt(i - 1)) {
                    dp[i][j] = dp[i - 1][j - 1];
                } else if (p.charAt(j - 1) == '*') {
                    dp[i][j] = dp[i - 1][j] || dp[i][j - 1];
                }
            }
        }
        return dp[m][n];
    }
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String s = sc.next();
    String p = sc.next();
    System.out.println(isMatch(s, p));
    sc.close();
 }
}
```

3. Output :



```
aa
a
false


** Process exited - Return Code: 0 **
```