FAST LEARNER COMPLEX PROBLEMS

PBLJ LAB:

Student Name: Shreejesh Prasad Patel **UID:** 22BCS50151

Section/Group: IOT-642-B **Branch:** BE-CSE

Date :08/04/25 **Subject Code:** 22CSH-359

1. Problem 1:

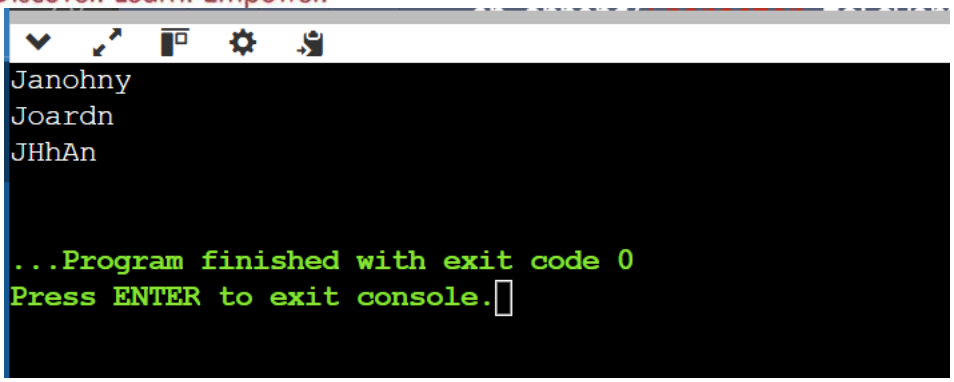
Encoding Three Strings: Anand was assigned the task of coming up with an encoding mechanism for any given three strings.He has come up with the following plan.

2.Implementation/Code:

```
public class StringEncoder {
  public static void main(String[] args) {
     String input1 = "John";
     String input2 = "Johny";
     String input3 = "Janardhan";

     String[] encoded = encodeStrings(input1, input2, input3);
     for (String s : encoded) {
        System.out.println(s);
     }
  }

  public static String[] encodeStrings(String a, String b, String c) {
     String[] splitA = splitThreeParts(a);
     String[] splitB = splitThreeParts(b);
     String[] splitC = splitThreeParts(c);

     // Combine parts to form three encoded strings
     String output1 = splitC[0] + splitA[1] + splitB[2];
     String output2 = splitB[0] + splitC[1] + splitA[2];
     String output3 = splitA[0] + splitB[1] + splitC[2];

     // Toggle alternate characters in output3
     output3 = toggleAlternateCase(output3);

     return new String[]{output1, output2, output3};
  }

  // Split string into 3 parts based on length and remainder
  public static String[] splitThreeParts(String s) {
     int len = s.length();
```

```
        int part = len / 3;
        int rem = len % 3;

        int front = part;
        int mid = part;
        int end = part;

        if (rem == 1) {
            mid++;
        } else if (rem == 2) {
            front++;
            end++;
        }

        String[] parts = new String[3];
        parts[0] = s.substring(0, front);
        parts[1] = s.substring(front, front + mid);
        parts[2] = s.substring(front + mid);

        return parts;
    }

    // Toggle the case of every alternate character starting from index 1
    public static String toggleAlternateCase(String s) {
        StringBuilder sb = new StringBuilder();

        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            if (i % 2 == 0) {
                sb.append(ch); // keep original
            } else {
                // toggle case
                if (Character.isLowerCase(ch)) {
                    sb.append(Character.toUpperCase(ch));
                } else {
                    sb.append(Character.toLowerCase(ch));
                }
            }
        }

        return sb.toString();
    }
}
```

2. Output:


Janohny
Joardn
JHhAn
...Program finished with exit code 0
Press ENTER to exit console.

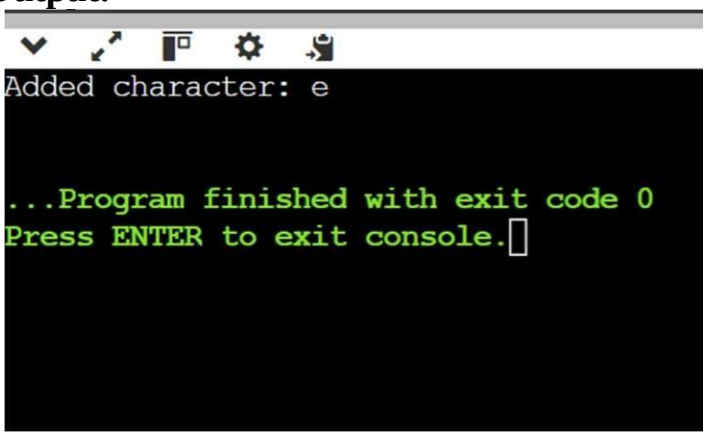
1. **Problem 2:**

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/Code:**

```
public class ExtraCharacterFinder {
    public static char findExtraChar(String s, String t) {
        int sum = 0;
        for (char ch : t.toCharArray()) sum += ch;
        for (char ch : s.toCharArray()) sum -= ch;
        return (char) sum;
    }

    public static void main(String[] args) {
        String s = "abcd";
        String t = "abcde";
        System.out.println("Added character: " + findExtraChar(s, t)); // Output: e
    }
}
```

3. **Output:**



1. **Problem 3:**

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

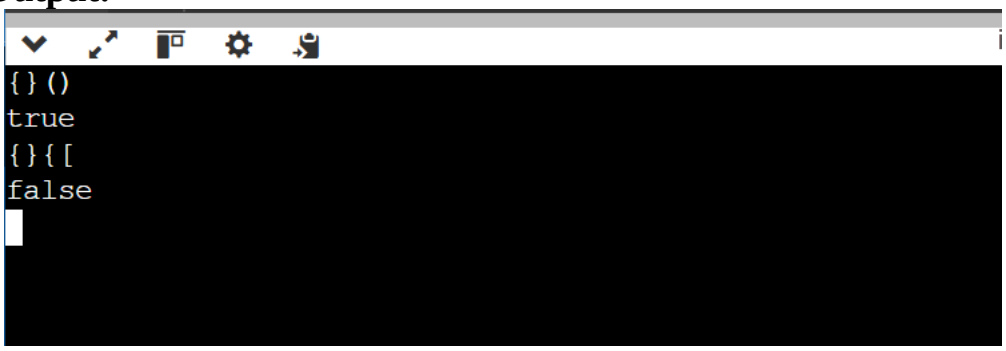
2. **Implementation/Code:**

```
import java.util.*;

public class BalancedParentheses {

    public static boolean isBalanced(String s) {
        Stack<Character> stack = new Stack<>();
        Map<Character, Character> matchingBracket = new HashMap<>();
        matchingBracket.put(')', '(');
        matchingBracket.put('}', '{');
        matchingBracket.put(']', '[');

        for (char c : s.toCharArray()) {
            if (matchingBracket.containsValue(c)) {
                stack.push(c);
            } else if (matchingBracket.containsKey(c)) {
                if (stack.isEmpty() || stack.peek() != matchingBracket.get(c)) {
                    return false;
                }
                stack.pop();
            }
        }

        return stack.isEmpty();
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        while (scanner.hasNextLine()) {
```

```
            String input = scanner.nextLine();
            System.out.println(isBalanced(input) ? "true" : "false");
        }

        scanner.close();
    }
}
```

```
}
```

3. **Output:**



1. **Problem 4:**

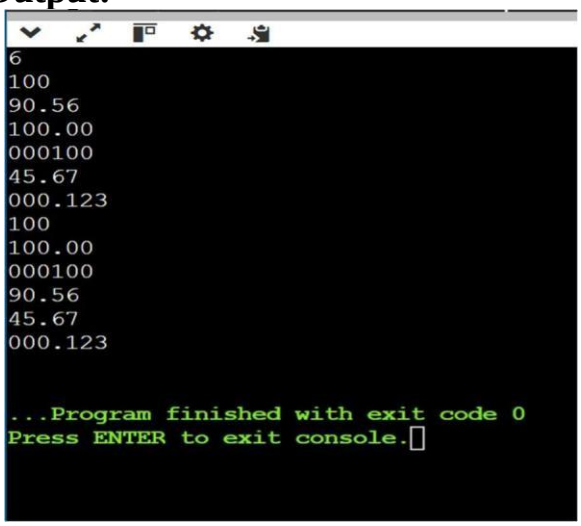
Java's BigDecimal class can handle arbitrary-precision signed decimal numbers. Let's test your knowledge of them! Given an array, , of real number strings, sort them in descending order — but wait, there's more! Each number must be printed in the exact same format as it was read from stdin, meaning that is printed as , and is printed as . If two numbers represent numerically equivalent values (e.g.,), then they must be listed in the same order as they were received as input). You must rearrange array 's elements according to the instructions above.

2. **Implementation/Code:**

```
import java.math.BigDecimal;
import java.util.*;

public class BigDecimalSort {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = Integer.parseInt(sc.nextLine());
        String[] values = new String[n];

        for (int i = 0; i < n; i++) {
            values[i] = sc.nextLine();
```

```
        }

        Arrays.sort(values, (a, b) -> {
            BigDecimal aVal = new BigDecimal(a);
            BigDecimal bVal = new BigDecimal(b);
            return bVal.compareTo(aVal); // descending
        });

        for (String v : values) {
            System.out.println(v);
        }
    }}
```

3. Output:



1. Problem 5:

Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value. If target is not found in the array, return [-1, -1]. You must write an algorithm with O(log n) runtime complexity.

2. Implementation/Code:

```
public class TargetRangeFinder {
    public static int[] searchRange(int[] nums, int target) {
        int start = findBound(nums, target, true);
        int end = findBound(nums, target, false);
```

```
        return new int[]{start, end};
    }

    private static int findBound(int[] nums, int target, boolean isFirst) {
        int low = 0, high = nums.length - 1, bound = -1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (nums[mid] == target) {
                bound = mid;
                if (isFirst) high = mid - 1;
                else low = mid + 1;
            } else if (nums[mid] < target) low = mid + 1;
            else high = mid - 1;
        }
        return bound;
    }

    public static void main(String[] args) {
        int[] nums = { 7, 6, 8, 8, 10,12};
        int target = 8;
        int[] result = searchRange(nums, target);
        System.out.println("[" + result[0] + ", " + result[1] + "]");
    }
}
```

3. Output: