



## FAST LEARNER COMPLEX PROBLEMS

### PBLJ LAB:

**Student Name:** Yubraj

**UID:** 22BCS17251

**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[] out = encodeStrings(input1, input2, input3);
        for (String s : out) {
            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);

        String output1 = splitA[0] + splitB[1] + splitC[2];
        String output2 = splitA[1] + splitB[2] + splitC[0];
        String output3 = splitA[2] + splitB[0] + splitC[1];
        output3 = toggleCase(output3);
    }
}
```

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

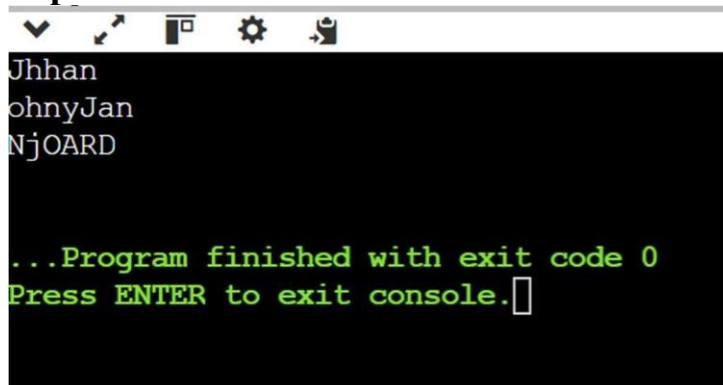
    public static String[] splitThreeParts(String s) {
        int len = s.length();
        int rem = len % 3;
        int part = len / 3;
        int front = part, mid = part, end = part;

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

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

    public static String toggleCase(String s) {
        StringBuilder sb = new StringBuilder();
        for (char ch : s.toCharArray()) {
            sb.append(Character.isLowerCase(ch) ? Character.toUpperCase(ch) :
Character.toLowerCase(ch));
        }
        return sb.toString();
    }
}
```

### 3. Output:



```
Jhhan
ohnyJan
NjOARD

...Program finished with exit code 0
Press ENTER to exit console.
```

## 1. Problem 2:

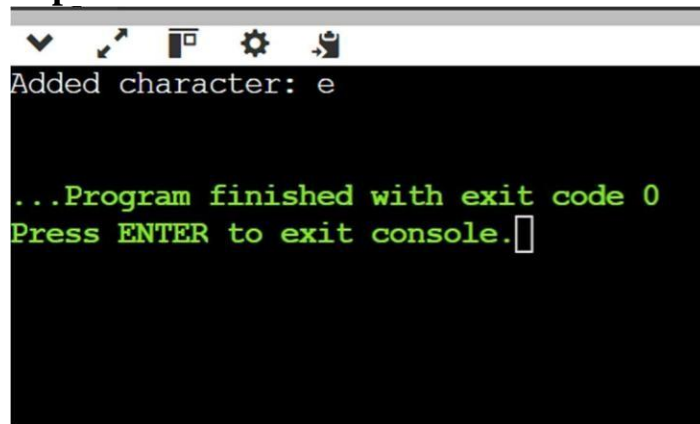
String t is generated by random shuffling string s and then add one more letter at a random position. Return the letter that was added to t.

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:



```
Added character: e  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

## 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 str) {
        Stack<Character> stack = new Stack<>();
        for (char ch : str.toCharArray()) {
            switch (ch) {
                case '(': case '{': case '[':
                    stack.push(ch); break;
                case ')':
                    if (stack.isEmpty() || stack.pop() != '(') return false;
                    break;
                case '}':
                    if (stack.isEmpty() || stack.pop() != '{') return false;
                    break;
                case ']':
                    if (stack.isEmpty() || stack.pop() != '[') return false;
                    break;
            }
        }
        return stack.isEmpty();
    }

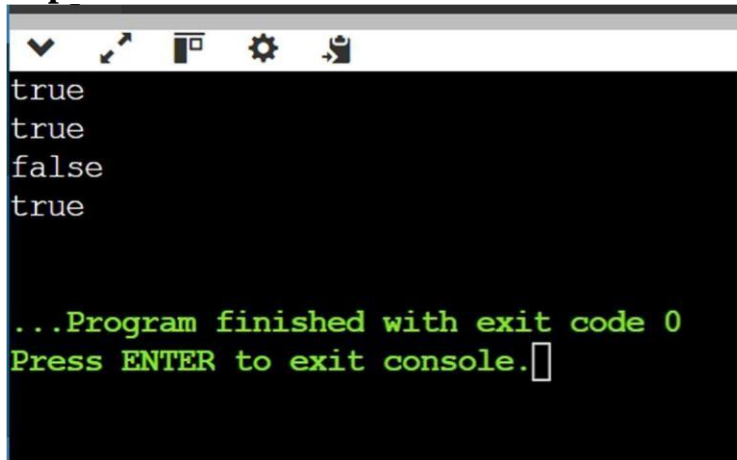
    public static void main(String[] args) {
        String[] inputs = {"{}()", "({()})", "{((", "[[]]"};
        for (String s : inputs) {
            System.out.println(isBalanced(s));
        }
    }
}
```



# **DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

Discover. Learn. Empower.

}

**3. Output:**

```
true
true
false
true

...Program finished with exit code 0
Press ENTER to exit console.
```

**1. Problem 4:**

Java's `BigDecimal` class can handle arbitrary-precision signed decimal numbers. Let's test your knowledge of them! Given an array, `arr`, 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 `1.0` is printed as `1.0`, and `1` is printed as `1`. If two numbers represent numerically equivalent values (e.g., `1.0` and `1`), then they must be listed in the same order as they were received as input). You must rearrange array `arr`'s elements according to the instructions above.

**2. Implementation/Code:**

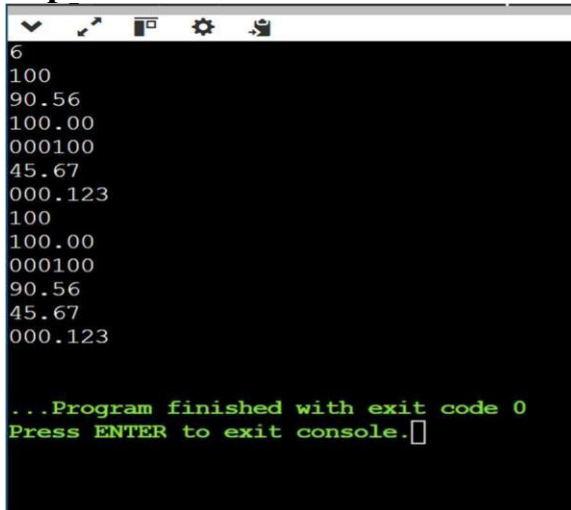
```
import java.math.BigDecimal;
import java.util.*;

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:



```
6  
100  
90.56  
100.00  
000100  
45.67  
000.123  
100  
100.00  
000100  
90.56  
45.67  
000.123  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

### 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:

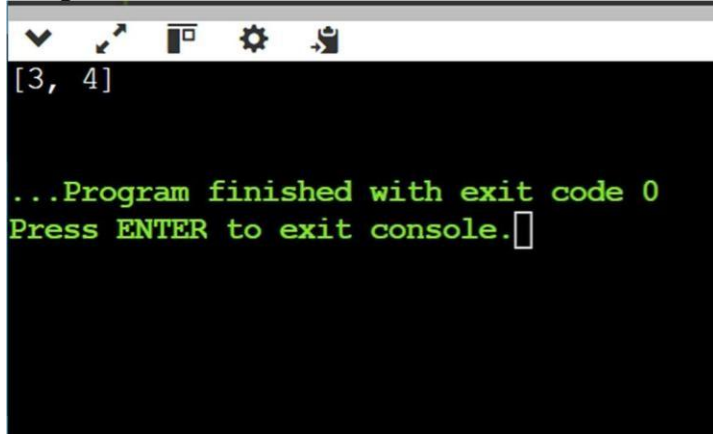
```
public class TargetRangeFinder {  
    public static int[] searchRange(int[] nums, int target) {  
        int start = findBound(nums, target, true);  
        int end = findBound(nums, target, false);
```

```
        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 = {5, 7, 7, 8, 8, 10};
        int target = 8;
        int[] result = searchRange(nums, target);
        System.out.println "[" + result[0] + ", " + result[1] + "]";
    }
}
```

### 3. Output:



```
[3, 4]

...Program finished with exit code 0
Press ENTER to exit console.
```