Department: BE-CSE/IT 3rd Year
Name: Shirsha Rakshit
UID:22BCS12279
Section:618-IOT-A
Subject: Project Based Learning in Java Subject Code: 22CSH-359/ 22ITH-359
Semester: 6th **Batch: 2022**

Lab Based Complex Coding Problems

Problem 1.

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.

Objective:

To process a paragraph of text by removing punctuation, normalizing case, and counting the frequency of each word. The program also displays the result in descending order of word frequency to analyze text data efficiently.

Code:

```
import java.util.*;

 public class WordFrequency {

    public static void main(String[] args) {

        String paragraph = "Java is a high-level programming language. Java is

 object-oriented. Java is widely used.";

        paragraph = paragraph.replaceAll("[.,!?]", "").toLowerCase();

        String[] words = paragraph.split(" ");

        Map<String, Integer> freqMap = new HashMap<>();

        for (String word : words) {
```

```
freqMap.put(word, freqMap.getOrDefault(word, 0) + 1);

    }


    List<Map.Entry<String, Integer>> sortedList = new


ArrayList<>(freqMap.entrySet());

    sortedList.sort((a, b) -> b.getValue() - a.getValue());

    for (Map.Entry<String, Integer> entry : sortedList) {

      System.out.println(entry.getKey() + ": " + entry.getValue());

    }

  }

}
```

Problem 3:

Given a String (In Uppercase alphabets or Lowercase alphabets), new alphabets is to be appended with following rule:

(i) If the alphabet is present in the input string, use the numeric value of that alphabet. E.g. a or A numeric value is 1 and so on. New alphabet to be appended between 2 alphabets:
   (a) If (sum of numeric value of 2 alphabets) %26 is 0, then append 0. E.g. string is ay. Numeric value of a is 1, y is 25. Sum is 26. Remainder is 0, the new string will be a0y.
   (b) Otherwise (sum of numeric value of 2 alphabets) %26 numeric value alphabet is to be appended. E.g. ac is string. Numeric value of a is 1, c is 3, sum is 4. Remainder with 26 is 4. Alphabet to be appended is d. output will be adc.

(ii) If a digit is present, it will be the same in the output string. E.g. string is 12, output string is 12.

(iii) If only a single alphabet is present, it will be the same in the output string. E.g. input string is 1a, output will be 1a.

(iv) If space is present, it will be the same in the output string. E.g. string is ac 12a, output will be adc 12a.

Constraint: Whether string alphabets are In Uppercase or Lowercase, appended alphabets must be in lower case. Output string must also be in lowercase.

Objective:
To read the contents of a text file and determine the total number of characters, words, and lines in it. This helps understand basic file handling and text parsing in Java.

Code:

```
import java.io.*;
public class FileReadStats {
   public static void main(String[] args) {
       String fileName = "sample.txt";
       int charCount = 0;
       int wordCount = 0;
       int lineCount = 0;
       try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {



           String line;
           while ((line = reader.readLine()) != null) {
               lineCount++;
               charCount += line.length();
               String[] words = line.trim().split("\s+");
               if (!line.trim().isEmpty()) {
                   wordCount += words.length;
               }
           }
           System.out.println("Characters: " + charCount);
           System.out.println("Words: " + wordCount);
           System.out.println("Lines: " + lineCount);
       } catch (IOException e) {
           System.out.println("File not found or error reading file.");
           e.printStackTrace();
       }
   }
```

Problem 5:

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 = [1,3,4,2]. There is no next greater element, so the answer is -1.
- 1 is underlined in nums2 = [1,3,4,2]. The next greater element is 3.
- 2 is underlined in nums2 = [1,3,4,2]. There is no next greater element, so the answer is -1.

Objective:
To reverse a given string using a stack data structure, demonstrating LIFO (Last In First Out) behavior in Java.

Code:

```
import java.util.*;
 public class StringReversalStack {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String input = sc.nextLine();
        Stack<Character> stack = new Stack<>();
        for (char ch : input.toCharArray()) {
            stack.push(ch);
        }
        StringBuilder reversed = new StringBuilder();
        while (!stack.isEmpty()) {
            reversed.append(stack.pop());
        }
        System.out.println("Reversed string: " + reversed.toString());
    }
```

Problem 7:

Comparators are used to compare two objects. In this challenge, you'll create a comparator and use it to sort an array.
The Player class has fields: a String and a integer.
Given an array of Player objects, write a comparator that sorts them in order of decreasing score; if or more players have the same score, sort those players alphabetically by name.

To do this, you must create a Checker class that implements the Comparator interface, then write an int compare(Player a, Player b) method implementing the Comparator.compare(T o1, T o2) method.
Input Format
The first line contains an integer, denoting the number of players.
Each of the subsequent lines contains a player's and , respectively.
Constraints

- players can have the same name.
- Player names consist of lowercase English letters.

Sample Input
5
amy 100
david 100
heraldo 50
aakansha 75
aleksa 150
Sample Output
aleksa 150
amy 100
david 100
aakansha 75
heraldo 50

Objective:
To create and run multiple threads using the Runnable interface and demonstrate concurrent execution of tasks in Java.

Code:
```
class PrintTask implements Runnable {
   private String message;
   PrintTask(String message) {
      this.message = message;
   }
   public void run() {
      for (int i = 1; i <= 5; i++) {
         System.out.println(message + " - Count: " + i);
         try {
            Thread.sleep(500);
         } catch (InterruptedException e) {
            System.out.println("Thread interrupted.");
         }
      }
```

```
    }
  }
  public class MultiThreadExample {
    public static void main(String[] args) {
      Thread t1 = new Thread(new PrintTask("Thread 1"));
      Thread t2 = new Thread(new PrintTask("Thread 2"));
      t1.start();
      t2.start();
    }
  }
```

Problem 9:

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 '*'.

Objective:
To create a graphical user interface (GUI) using Swing that converts temperature from Celsius to Fahrenheit, showcasing Javas GUI-building capabilities.

Code:

```
import javax.swing.*;
import java.awt.event.*;
public class TempConverter {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Celsius to Fahrenheit");
    JTextField celsiusField = new JTextField();
    JLabel resultLabel = new JLabel("Fahrenheit: ");
    JButton convertBtn = new JButton("Convert");
    celsiusField.setBounds(50, 50, 150, 20);
    convertBtn.setBounds(50, 100, 100, 30);
    resultLabel.setBounds(50, 150, 200, 30);
    convertBtn.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        try {
          double celsius = Double.parseDouble(celsiusField.getText());
          double fahrenheit = (celsius * 9/5) + 32;
```

```
                        resultLabel.setText("Fahrenheit: " + fahrenheit);
                    } catch (NumberFormatException ex) {
                        resultLabel.setText("Please enter a valid number.");
                    }
                }
            });
            frame.add(celsiusField);
            frame.add(convertBtn);
            frame.add(resultLabel);
            frame.setSize(300, 300);
            frame.setLayout(null);
            frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
```

LEARNING OUTCOMES:

1. String Manipulation: Learn how to process strings by removing punctuation, converting to lowercase, and splitting words for frequency analysis, helping to enhance text processing skills.
2. File Handling: Gain practical experience in reading from and writing to files, including performing character, word, and line counting, while handling file I/O exceptions.
3. Basic Arithmetic Operations: Understand how to implement a basic calculator using switch-case statements for different mathematical operations, reinforcing control flow concepts.
4. Multithreading: Learn how to implement multi-threaded applications for concurrent tasks, such as printing even and odd numbers, and understand the use of thread synchronization with sleep methods.
5. Method Overloading: Practice method overloading by defining multiple methods with the same name but different parameters to calculate the areas of various shapes, showcasing polymorphism in Java.

4o mini