2300031099

N.LAKSHMI SOWJANYA

Skill week 6

1) Given a string s, regroup the characters of s so that any two adjacent characters are not the same. Return any possible rearrangement of s or return ' if not possible.

Details:

a) Input: A string s consisting of lowercase English letters. The length of s is between 1 and 500 characters.

b) Output: A string where no two adjacent characters are identical. If such a rearrangement is not possible, return an empty string. Example:

a) Input: s = "aab 11

Output: "aba" (or any other valid rearrangement like "baa")

b) Input: s = "aaab"

Output: (since it's not possible to rearrange the characters without having at least two adjacent as) Constraints:

a) The string length is between 1 and 500 characters.

b) The string consists only of lowercase English letters.

Approach:

v Character Frequency with LinkedList: Use a LinkedList to store characters and their frequencies, ensuring you can maintain and update frequencies as needed.

V PriorityQueue (Min-Heap): Use the PriorityQueue with a custom comparator to manage the characters based on their frequencies.

V Stack for Previous Characters: Use a Stack to track previously placed characters to avoid placing the same character consecutively.

```
package skill61;


import java.util.HashMap;

import java.util.LinkedList;

import java.util.Map;

import java.util.PriorityQueue;

import java.util.Queue;


class Solution {
```

```
public String reorganizeString(String s) {
    if (s == null || s.length() == 0) {
        return "";
    }
    Map<Character, Integer> frequencyMap = new HashMap<>();
    for (char c : s.toCharArray()) {
        frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);
    }
    PriorityQueue<Map.Entry<Character, Integer>> maxHeap = new PriorityQueue<>(
        (a, b) -> b.getValue() - a.getValue()
    );
    maxHeap.addAll(frequencyMap.entrySet());
    StringBuilder result = new StringBuilder();
    Queue<Map.Entry<Character, Integer>> waitQueue = new LinkedList<>();

    while (!maxHeap.isEmpty()) {
        Map.Entry<Character, Integer> current = maxHeap.poll();
        result.append(current.getKey());
        current.setValue(current.getValue() - 1);
        waitQueue.offer(current);
        if (waitQueue.size() > 1) {
            Map.Entry<Character, Integer> front = waitQueue.poll();
            if (front.getValue() > 0) {
                maxHeap.offer(front);
            }
        }
    }
    return result.length() == s.length() ? result.toString() : "";
}

public static void main(String[] args) {
```

```
        Solution solution = new Solution();

        System.out.println(solution.reorganizeString("aab"));
        System.out.println(solution.reorganizeString("aaab"));
    }
}

package skill61;

public class Main {
            public static void main(String[] args) {
        Solution solution = new Solution();
        String s1 = "aab";
        String result1 = solution.reorganizeString(s1);
        System.out.println("Input: " + s1 + " -> Output: " + result1);
        String s2 = "aaab";
        String result2 = solution.reorganizeString(s2);
        System.out.println("Input: " + s2 + " -> Output: " + result2);
        String s3 = "vvvlo";
        String result3 = solution.reorganizeString(s3);
        System.out.println("Input: " + s3 + " -> Output: " + result3);

        String s4 = "a";
        String result4 = solution.reorganizeString(s4);
        System.out.println("Input: " + s4 + " -> Output: " + result4);

        String s5 = "ab";
        String result5 = solution.reorganizeString(s5);
        System.out.println("Input: " + s5 + " -> Output: " + result5);
    }
}
```

```
Problems  @ Javadoc  Declaration  Console ×
<terminated> Solution [Java Application] C:\Users\laksh\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.
aba
```

2) Advanced Type Bounds with Generics

Objective: Create a generic class that uses advanced type bounds and wildcards and understand how the generic methods work with type bounds and wildcards.

Details:

Class Definition:

a) Define T with multiple bounds: it must implement both Comparable<T>.

Methods:

a) processList(List<? extends T> list): This method should process a list of elements that are of type T or its subtypes. It should iterate through the list and print each element.

b) addToList(List<? super T> list, T element): This method should add an element of type T to a list that can hold T or any supertype of T.

Use Case:

a) Create a Product class that implements Comparable<Product> and Serializable.

b) Use AdvancedGeneric with Product to:

v/ Process a list of Product objects.

Add a new Product to another list and process it.

```
package skill62;

import java.io.Serializable;
import java.util.List;

public class AdvancedGeneric<T extends Comparable<T> & Serializable> {
    public void processList(List<? extends T> list) {
        for (T element : list) {
            System.out.println(element);
```

```
        }
    }
    public void addToList(List<? super T> list, T element) {
        list.add(element);
    }
}

package skill62;

import java.io.Serializable;

public class Product implements Comparable<Product>, Serializable {
    private String name;
    private double price;

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    @Override
    public int compareTo(Product other) {
        return Double.compare(this.price, other.price);
    }

    @Override
    public String toString() {
        return "Product{name='" + name + "', price=" + price + "}";
    }
}
```

```java
package skill62;

import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        AdvancedGeneric<Product> advancedGeneric = new AdvancedGeneric<>();
        List<Product> productList = new ArrayList<>();
        productList.add(new Product("Laptop", 999.99));
        productList.add(new Product("Smartphone", 599.99));
        productList.add(new Product("Tablet", 299.99));
        System.out.println("Processing Product List:");
        advancedGeneric.processList(productList);
        List<Object> objectList = new ArrayList<>();
        Product newProduct = new Product("Smartwatch", 199.99);
        advancedGeneric.addToList(objectList, newProduct);
        System.out.println("Processing Updated List:");
        for (Object obj : objectList) {
            System.out.println(obj);
        }
    }
}
```

Problems Javadoc Declaration Console ×

<terminated> Main (12) [Java Application] C:\Users\laksh\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre

```
Processing Product List:
Product{name='Laptop', price=999.99}
Product{name='Smartphone', price=599.99}
Product{name='Tablet', price=299.99}
Processing Updated List:
Product{name='Smartwatch', price=199.99}
```

3) A class Employee that has the following requirements:

Objective: To Understand the functionality of a custom Comparator for sorting Employee objects by multiple criteria and to ensure correct application of the complex sorting logic.

Class Definition:

a) The Employee class has the following attributes:

name: String, v/ age: Integer, v/ Salary: Double.

b) Implement the Comparable<Employee> interface in the Employee class to provide a natural ordering based on salary.

Custom Comparator Implementation:

a) Implement a custom Comparator<Employee> that provides the following functionalities:

v/ Primary Sorting: Sort employees by age in ascending order.

v/ Secondary Sorting: If two employees have the same age, sort them by salary in descending order.

v/ Tertiary Sorting: If two employees have the same age and salary, sort them by name

```
package skill63;


import java.util.Comparator;


public class EmployeeComparator implements Comparator<Employee> {
   @Override
   public int compare(Employee e1, Employee e2) {


      int ageComparison = e1.getAge().compareTo(e2.getAge());
      if (ageComparison != 0) {
         return ageComparison;
      }



      int salaryComparison = e2.getSalary().compareTo(e1.getSalary());
      if (salaryComparison != 0) {
         return salaryComparison;
      }
```

```
        return e1.getName().compareTo(e2.getName());
    }
}
package skill63;

public class Employee implements Comparable<Employee> {
    private String name;
    private Integer age;
    private Double salary;

    public Employee(String name, Integer age, Double salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public Integer getAge() {
        return age;
    }

    public Double getSalary() {
        return salary;
    }

    @Override
```

```
    public int compareTo(Employee other) {
        return this.salary.compareTo(other.salary);
    }

    @Override
    public String toString() {
        return "Employee{name='" + name + "', age=" + age + ", salary=" + salary + "}";
    }
}
package skill63;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Alice", 30, 70000.0));
        employees.add(new Employee("Bob", 25, 50000.0));
        employees.add(new Employee("Charlie", 30, 50000.0));
        employees.add(new Employee("David", 25, 60000.0));
        employees.add(new Employee("Eve", 30, 70000.0));

        Collections.sort(employees, new EmployeeComparator());



        for (Employee employee : employees) {
            System.out.println(employee);
        }
    }
```

}

Problems Javadoc Declaration Console

<terminated> Main (13) [Java Application] C:\Users\laksh\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\b

```
Employee{name='David', age=25, salary=60000.0}
Employee{name='Bob', age=25, salary=50000.0}
Employee{name='Alice', age=30, salary=70000.0}
Employee{name='Eve', age=30, salary=70000.0}
Employee{name='Charlie', age=30, salary=50000.0}
```