Candidato: Ing. Alex Tumiri Huanca

Fecha: 08-24-2024

Puesto: Programador Fullstack

Link del Repositorio de retos resueltos: [*https://github.com/Zelechos/Prueba-tecnica-Programador-Fullstack-Efiempresa*](https://github.com/Zelechos/Prueba-tecnica-Programador-Fullstack-Efiempresa)

**Reto 1: Determinar si una nota puede ser escrita con el texto de una revista**

**Tiempo de Resolución:** ***45 Minutos***

**Descripción del Problema:**

Se nos pide escribir una función que determine si es posible escribir una nota usando los recortes de una revista. La función debe recibir dos cadenas de caracteres: la primera es la nota que deseamos escribir, y la segunda es el texto completo de la revista. La función debe devolver true si la nota puede ser escrita usando solo las palabras de la revista, y false en caso contrario.

**Análisis:**

- Para determinar si la nota puede ser escrita usando la revista, necesitamos asegurarnos de que cada palabra de la nota esté disponible en la revista con frecuencia.

- Un enfoque eficiente para este problema es utilizar un mapa (hashmap) para contar la frecuencia de cada palabra en la revista y luego verificar si cada palabra en la nota puede ser satisfecha con el conteo de la revista.

**Complejidad:**

- Si ***n*** es el número de palabras en la nota y ***m*** es el número de palabras en la revista, la complejidad será ***O(n + m)***. Esto se debe a que necesitamos recorrer ambas cadenas una vez para contar y verificar las palabras.

**Solución en Código:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | package org.example.first;  import java.util.Arrays;  import java.util.HashMap;  import java.util.List;  import java.util.Map;  /\*\*  \* Class that implements the logic to check if it is possible to write a note  \* using the words available in a magazine.  \*/  public class FirstChallenge {  /\*\*  \* Check if it is possible to write the note using the words available in the magazine.  \*  \* @param note The note you want to write.  \* @param magazine The magazine with the available words.  \* @return true if it is possible to write the note, false otherwise.  \*/  public boolean canWriteNote(String note, String magazine) {  // Validation if the note or magazine does not have words  if (note.isEmpty() || magazine.isEmpty()) return false;  // Separate the words of the note and the magazine  List<String> wordsNote = Arrays.stream(note.split("\\s+")).toList();  List<String> wordsMagazine = Arrays.stream(magazine.split("\\s+")).toList();  // Create a map to count the words in the magazine  Map<String, Integer> countMagazine = new HashMap<>();  // Count the frequency of each word in the magazine  wordsMagazine.forEach(word -> countMagazine.put(word, countMagazine.getOrDefault(word, 0) + 1));  // Check if the note can be built with the magazine  return wordsNote.stream().allMatch(word -> {  int count = countMagazine.getOrDefault(word, 0);  if (count == 0) return false; // No hay suficientes palabras, no se puede escribir la nota  countMagazine.put(word, count - 1);  return true;  });  }  } |

**Breve Explicación:**

Primero se Verifica si la nota o la revista están vacías. Si cualquiera de las dos está vacío, devuelve false, ya que no es posible escribir la nota sin palabras disponibles, posteriormente separo las palabras de la nota y de la revista en listas utilizando el método **split("\\s+")**, que divide las cadenas en palabras basándose en espacios en blanco y creo un mapa (countMagazine) para contar la frecuencia de cada palabra en la revista, entonces para cada palabra en la revista, incrementa su cuenta en el mapa. Si la palabra no existe en el mapa, la agrega con una cuenta inicial de 1 entonces recorremos cada palabra de la nota y verificamos si está disponible en la revista con suficiente frecuencia, si la palabra está disponible, reduce su cuenta en el mapa, si alguna palabra de la nota no está disponible o no hay suficientes ocurrencias, devuelve **false** y por último si todas las palabras de la nota pueden ser satisfechas con las palabras de la revista, devuelve **true**.

**Test Unitarios:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | package org.example.first;  import org.junit.Test;  import static org.junit.Assert.assertFalse;  import static org.junit.Assert.assertTrue;  /\*\*  \* Tests for the First Challenge class.  \*/  public class FirstChallengeTest {  @Test  public void testCanWriteNoteSuccessCase() {  // Arrange  FirstChallenge firstChallenge = new FirstChallenge();  // Act & Assert  assertTrue(firstChallenge.canWriteNote("es un informe", "esto es un simple informe de prueba"));  }  @Test  public void testCanWriteNoteFailedCase() {  // Arrange  FirstChallenge firstChallenge = new FirstChallenge();  // Act & Assert  assertFalse(firstChallenge.canWriteNote("esto es una prueba", "esto es solo una simple"));  }  @Test  public void testCanWriteNoteCaseEmpty() {  // Arrange  FirstChallenge firstChallenge = new FirstChallenge();  // Act & Assert  assertFalse(firstChallenge.canWriteNote("", "cualquier cosa"));  assertFalse(firstChallenge.canWriteNote("cualquier cosa", ""));  }  } |

**Reto 2: Notificación de estudiantes afectados por una incidencia**

**Tiempo de Resolución:** ***40 Minutos***

**Descripción del Problema:**

Tenemos listas de estudiantes matriculados en dos asignaturas diferentes (Matemáticas y Francés) y una lista de estudiantes matriculados en ambas asignaturas. Necesitamos notificar a los estudiantes afectados por una incidencia en la sede de Málaga, asegurándonos de que cada estudiante reciba solo un email.

**Análisis:**

- Se nos proporciona una lista de estudiantes matriculados en Matemáticas, Francés y ambas. Debemos filtrar aquellos que asisten a la sede de Málaga.

- Además, debemos asegurar que cada estudiante reciba solo un email, por lo que debemos eliminar duplicados y enviar el correo solo una vez.

**Complejidad:**

- La complejidad principal estará en la combinación y filtrado de listas. Si ***n*** es el total de estudiantes en las listas, la complejidad será ***O(n)***.

**Solución en Código:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | package org.example.second;  import java.util.HashSet;  import java.util.List;  import java.util.Set;  import java.util.stream.Collectors;  /\*\*  \* Class that implements the logic to obtain all the students from Malaga  \* enrolled in Mathematics and French  \* to send the respective emails  \*/  public class SecondChallenge {  /\*\*  \* Notifies students based on their headquarters.  \*  \* @param listA List of students from subject A.  \* @param listB List of students from subject B.  \* @param listC List of students from both subjects A and B.  \* @param headquarters The headquarters (sede) to filter students.  \* @return Set of student names from Malaga.  \*/  public Set<String> notifyStudents(  List<String> listA,  List<String> listB,  List<String> listC,  String headquarters) {  // Combine all student names from the three lists  Set<String> allStudents = new HashSet<>();  allStudents.addAll(listA);  allStudents.addAll(listB);  allStudents.addAll(listC);  // Filter students from Malaga to send Emails  return allStudents.stream()  .filter(student -> student.contains(headquarters))  .collect(Collectors.toSet());  }  } |

**Explicación:**

Implementamos la lógica para obtener los nombres de los estudiantes de la sede de Málaga que están matriculados en Matemáticas y Francés, con el objetivo de enviarles los respectivos correos electrónicos, creamos las listas A, B, C y tambien un parámetro headquartes mediante el cual obtendremos la sede, Creamos un conjunto (Set) llamado allStudents para combinar los nombres de los estudiantes de las tres listas (listA, listB, listC), asegurando que no haya duplicados y posteriormente se utilizó un flujo (stream) para filtrar a los estudiantes que pertenecen a la sede especificada (Málaga). Esto se realiza verificando si el nombre del estudiante contiene el nombre de la sede como resultado tenemos un conjunto (Set) con los nombres de los estudiantes que pertenecen a la sede de Málaga y que deben recibir los correos electrónicos.

**Test Unitarios:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88 | package org.example.second;  import org.junit.Test;  import java.util.Arrays;  import java.util.List;  import java.util.Set;  import static org.junit.Assert.\*;  /\*\*  \* Tests for the Second Challenge class.  \*/  public class SecondChallengeTest {  @Test  public void testNotifyStudents() {  // Arrange  SecondChallenge secondChallenge = new SecondChallenge();  String headquarters = "Málaga";  // Prepare sample student lists  List<String> listA = Arrays.asList("Ana Málaga", "Luis Madrid", "Pedro Málaga");  List<String> listB = Arrays.asList("Luis Málaga", "Sofia Málaga", "Ana Madrid");  List<String> listC = Arrays.asList("Pedro Málaga", "Luis Málaga", "Sofia Madrid");  // Act  Set<String> result = secondChallenge.notifyStudents(listA, listB, listC, headquarters);  // Assert  assertEquals(4, result.size()); // Expecting 4 students in total  assertTrue(result.contains("Ana Málaga")); // Ana is from Malaga  assertTrue(result.contains("Pedro Málaga")); // Pedro is from Malaga  assertTrue(result.contains("Luis Málaga")); // Luis is from Malaga  }  @Test  public void testNotifyStudentsSingleList() {  // Arrange  List<String> listA = List.of("Juan Málaga", "María Málaga", "Pedro Barcelona");  String headquarters = "Málaga";  SecondChallenge secondChallenge = new SecondChallenge();  // Act  Set<String> result = secondChallenge.notifyStudents(listA, List.of(), List.of(), headquarters);  // Assert  assertTrue(result.contains("Juan Málaga")); // Juan is from Málaga  assertTrue(result.contains("María Málaga")); // María is from Málaga  assertFalse(result.contains("Pedro Barcelona")); // Pedro is not from Málaga  }  @Test  public void testNotifyStudentsTwoLists() {  // Arrange  List<String> listA = List.of("Juan Málaga", "María Málaga");  List<String> listB = List.of("María Málaga", "Carlos Madrid");  String headquarters = "Málaga";  SecondChallenge secondChallenge = new SecondChallenge();  // Act  Set<String> result = secondChallenge.notifyStudents(listA, listB, List.of(), headquarters);  // Assert  assertTrue(result.contains("Juan Málaga")); // Juan is from Malaga  assertTrue(result.contains("María Málaga")); // María is from Malaga  assertFalse(result.contains("Carlos Madrid")); // Carlos is not from Malaga  }  @Test  public void testNoDuplicatesInNotification() {  // Arrange  List<String> listA = List.of("Juan Madrid", "María Málaga", "María Barcelona");  List<String> listB = List.of("María Málaga", "Carlos Málaga");  List<String> listC = List.of("María Málaga", "Ana Madrid");  String headquarters = "Málaga";  SecondChallenge secondChallenge = new SecondChallenge();  // Act  Set<String> result = secondChallenge.notifyStudents(listA, listB, listC, headquarters);  // Assert  assertEquals(2, result.size()); // We hope there are no duplicates  }  } |