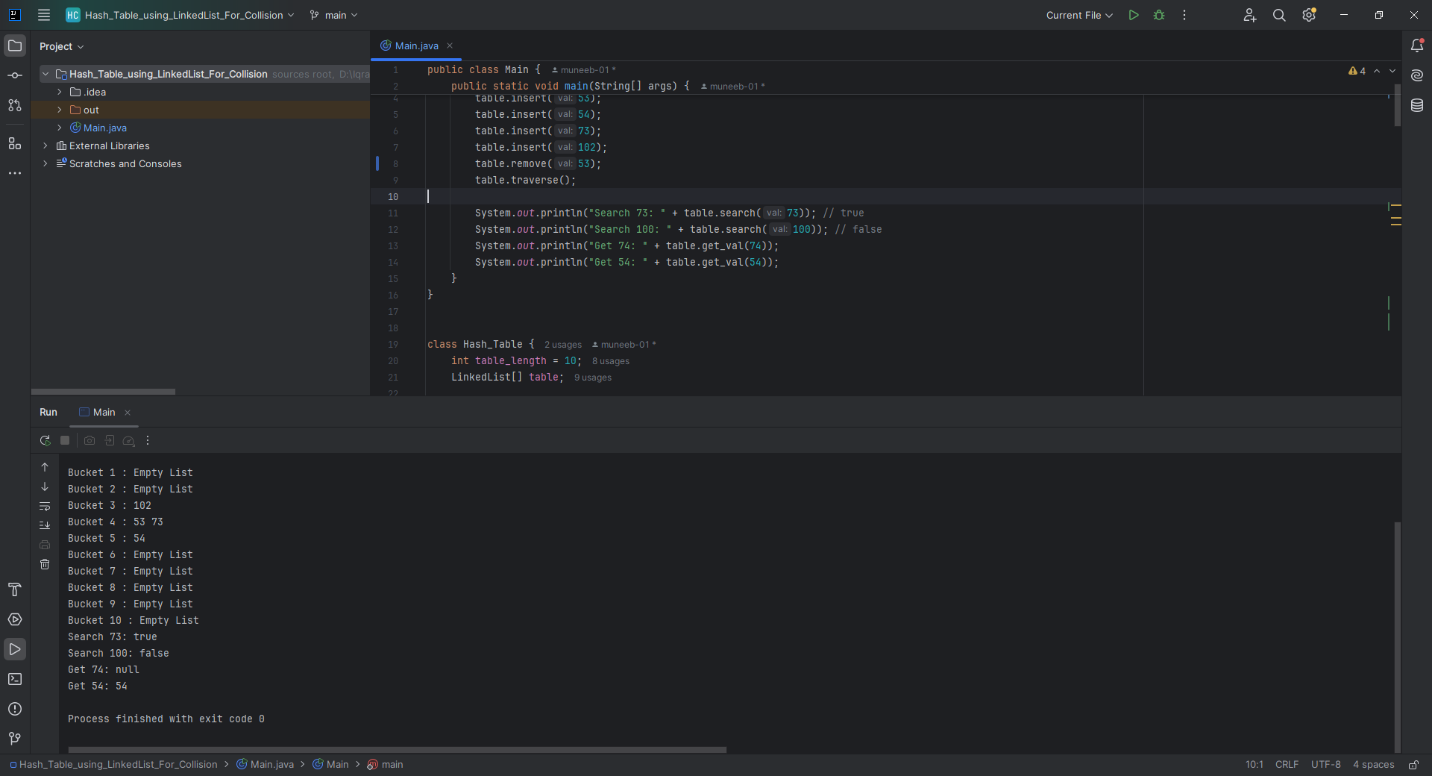
**CLASS TASK All:**

**CODE:**

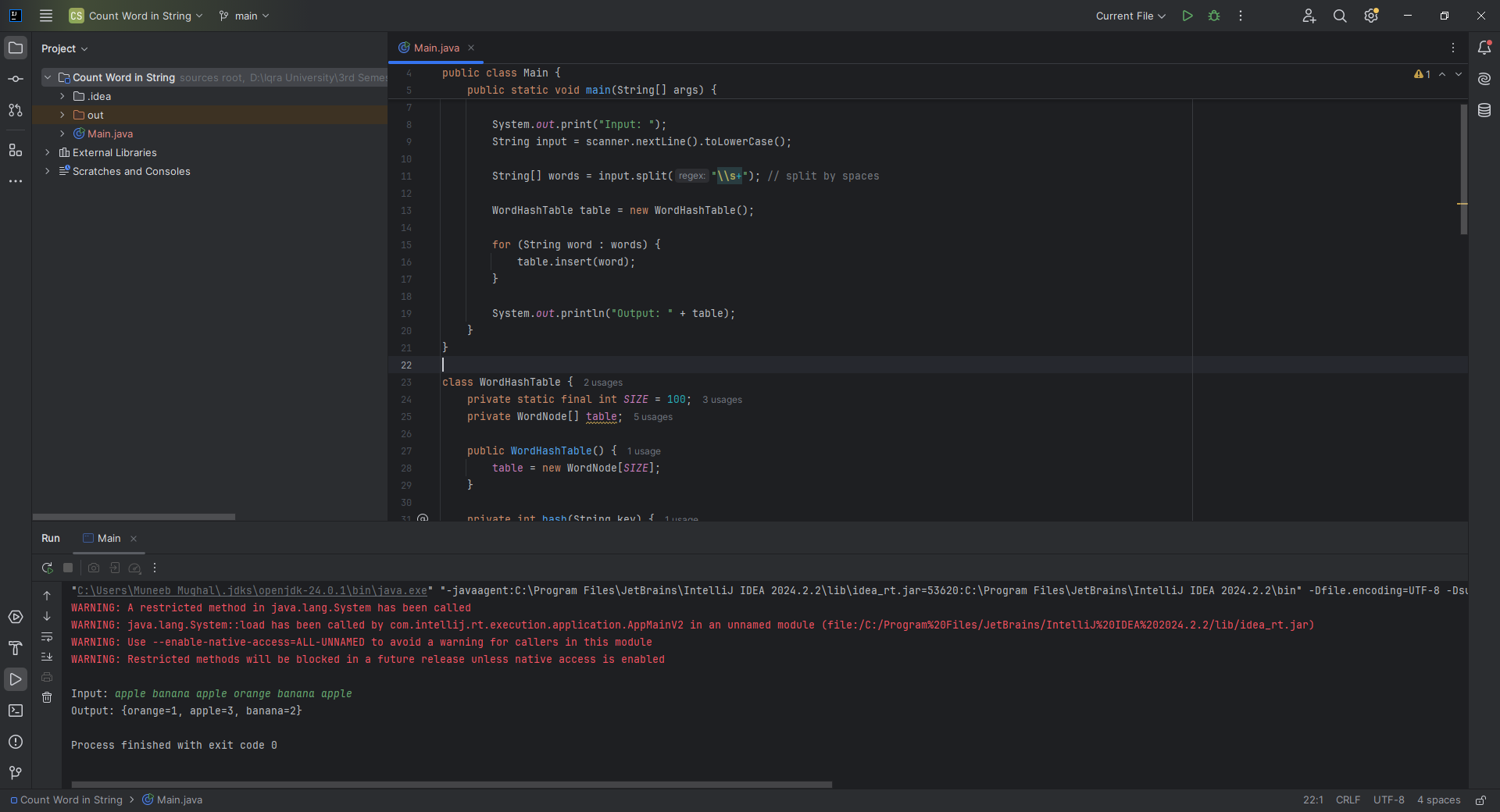
public class Main {  
 public static void main(String[] args) {  
 Hash\_Table table = new Hash\_Table();  
 table.insert(53);  
 table.insert(54);  
 table.insert(73);  
 table.insert(102);  
 table.remove(53);  
 table.traverse();  
  
 System.*out*.println("Search 73: " + table.search(73)); // true  
 System.*out*.println("Search 100: " + table.search(100)); // false  
 System.*out*.println("Get 74: " + table.get\_val(74));  
 System.*out*.println("Get 54: " + table.get\_val(54));  
 }  
}  
  
  
class Hash\_Table {  
 int table\_length = 10;  
 LinkedList[] table;  
  
 Hash\_Table(int table\_length){  
 this.table\_length = table\_length;  
 table = new LinkedList[table\_length];  
 for(int i = 0; i < table\_length; i++){  
 table[i] = new LinkedList();  
 }  
 }  
  
 Hash\_Table(){  
 table = new LinkedList[table\_length];  
 for(int i = 0; i < table\_length; i++){  
 table[i] = new LinkedList();  
 }  
 }  
  
 void insert(int val){  
 int key = val % table\_length;  
 table[key].insert\_at\_end(val);  
 }  
  
 boolean search(int val){  
 int key = val % table\_length;  
 return table[key].search(val);  
 }  
  
 void traverse(){  
 for(int i = 0; i < table\_length; i++){  
 System.*out*.print("Bucket " + (i+1) + " : ");  
 table[i].traverse();  
 }  
 }  
 Integer get\_val(int val){  
 int key = val % table\_length;  
 return table[key].get(val);  
 }  
  
 void remove(int val) {  
 int key = val % table\_length;  
 table[key].remove(val);  
 }  
  
  
 class LinkedList {  
 Node head = null;  
  
 class Node {  
 int data;  
 Node next;  
 public Node(int data) {  
 this.data = data;  
 next = null;  
 }  
 }  
  
 void insert\_at\_end(int data) {  
 Node newNode = new Node(data);  
 if(head == null){  
 head = newNode;  
 return;  
 }  
 Node current = head;  
 while(current.next != null){  
 current = current.next;  
 }  
 current.next = newNode;  
 }  
  
 void traverse(){  
 if(head == null){  
 System.*out*.println("Empty List");  
 return;  
 }  
 Node current = head;  
 while(current != null){  
 System.*out*.print(current.data + " ");  
 current = current.next;  
 }  
 System.*out*.println();  
 }  
 void remove(int val) {  
 if (head == null) return;  
  
 if (head.data == val) {  
 head = head.next;  
 return;  
 }  
  
 Node current = head;  
 while (current.next != null && current.next.data != val) {  
 current = current.next;  
 }  
  
 if (current.next != null) {  
 current.next = current.next.next;  
 }  
 }  
 // 🔍 New method for searching a value in the linked list  
 boolean search(int val){  
 Node current = head;  
 while(current != null){  
 if(current.data == val){  
 return true;  
 }  
 current = current.next;  
 }  
 return false;  
 }  
 Integer get(int val){  
 Node current = head;  
 while(current != null){  
 if(current.data == val){  
 return current.data;  
 }  
 current = current.next;  
 }  
 return null; // value not found  
 }  
 }  
}



**Home Task 2:** Counting Word Frequencies Using a Hash Table

**CODE:**

import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.print("Input: ");  
 String input = scanner.nextLine().toLowerCase();  
  
 String[] words = input.split("\\s+"); // split by spaces  
  
 WordHashTable table = new WordHashTable();  
  
 for (String word : words) {  
 table.insert(word);  
 }  
  
 System.*out*.println("Output: " + table);  
 }  
}  
  
class WordHashTable {  
 private static final int *SIZE* = 100;  
 private WordNode[] table;  
  
 public WordHashTable() {  
 table = new WordNode[*SIZE*];  
 }  
  
 private int hash(String key) {  
 return Math.*abs*(key.hashCode()) % *SIZE*;  
 }  
  
 public void insert(String word) {  
 int index = hash(word);  
 WordNode current = table[index];  
  
 while (current != null) {  
 if (current.word.equals(word)) {  
 current.count++;  
 return;  
 }  
 current = current.next;  
 }  
  
 WordNode newNode = new WordNode(word);  
 newNode.next = table[index];  
 table[index] = newNode;  
 }  
  
 @Override  
 public String toString() {  
 StringBuilder sb = new StringBuilder("{");  
 boolean first = true;  
  
 for (int i = 0; i < *SIZE*; i++) {  
 WordNode current = table[i];  
 while (current != null) {  
 if (!first) sb.append(", ");  
 sb.append(current.word).append("=").append(current.count);  
 first = false;  
 current = current.next;  
 }  
 }  
  
 sb.append("}");  
 return sb.toString();  
 }  
  
 static class WordNode {  
 String word;  
 int count;  
 WordNode next;  
  
 public WordNode(String word) {  
 this.word = word;  
 this.count = 1;  
 this.next = null;  
 }  
 }  
}



**Home Task 3:** Implementing a Hash Table for Storing Student Records

**CODE:**

public class Main {  
 public static void main(String[] args) {  
 StudentHashTable studentTable = new StudentHashTable();  
  
 // Adding Students  
 studentTable.addStudent(new Student(101, "John", "A"));  
 studentTable.addStudent(new Student(201, "Alice", "B"));  
 studentTable.addStudent(new Student(103, "Bob", "C"));  
  
 // Retrieve a Student  
 Student student = studentTable.getStudent(101);  
 if (student != null) {  
 System.*out*.println("Retrieve Student: For ID 101, output Name: " + student.name + ", Grade: " + student.grade);  
 } else {  
 System.*out*.println("Student with ID 101 not found.");  
 }  
  
 // Remove a Student  
 studentTable.removeStudent(102);  
  
 // Try to retrieve removed student  
 Student removed = studentTable.getStudent(102);  
 System.*out*.println(removed == null ? "Student with ID 102 was removed." : "Removal failed.");  
 }  
}  
  
class Student {  
 int id;  
 String name;  
 String grade;  
  
 public Student(int id, String name, String grade) {  
 this.id = id;  
 this.name = name;  
 this.grade = grade;  
 }  
}  
  
class StudentHashTable {  
 private static final int *SIZE* = 100;  
 private Node[] table;  
  
 public StudentHashTable() {  
 table = new Node[*SIZE*];  
 }  
  
 private int hash(int id) {  
 return id % *SIZE*;  
 }  
  
 public void addStudent(Student student) {  
 int index = hash(student.id);  
 Node head = table[index];  
  
 // Check if ID already exists, update  
 Node current = head;  
 while (current != null) {  
 if (current.student.id == student.id) {  
 current.student = student; // Update existing  
 return;  
 }  
 current = current.next;  
 }  
  
 // Else insert at beginning  
 Node newNode = new Node(student);  
 newNode.next = head;  
 table[index] = newNode;  
  
 System.*out*.println("Add Student: ID: " + student.id + ", Name: " + student.name + ", Grade: " + student.grade);  
 }  
  
 public void removeStudent(int id) {  
 int index = hash(id);  
 Node current = table[index];  
 Node prev = null;  
  
 while (current != null) {  
 if (current.student.id == id) {  
 if (prev == null) {  
 table[index] = current.next;  
 } else {  
 prev.next = current.next;  
 }  
 return;  
 }  
 prev = current;  
 current = current.next;  
 }  
 }  
  
 public Student getStudent(int id) {  
 int index = hash(id);  
 Node current = table[index];  
  
 while (current != null) {  
 if (current.student.id == id) {  
 return current.student;  
 }  
 current = current.next;  
 }  
 return null;  
 }  
  
 private static class Node {  
 Student student;  
 Node next;  
  
 public Node(Student student) {  
 this.student = student;  
 this.next = null;  
 }  
 }  
}

