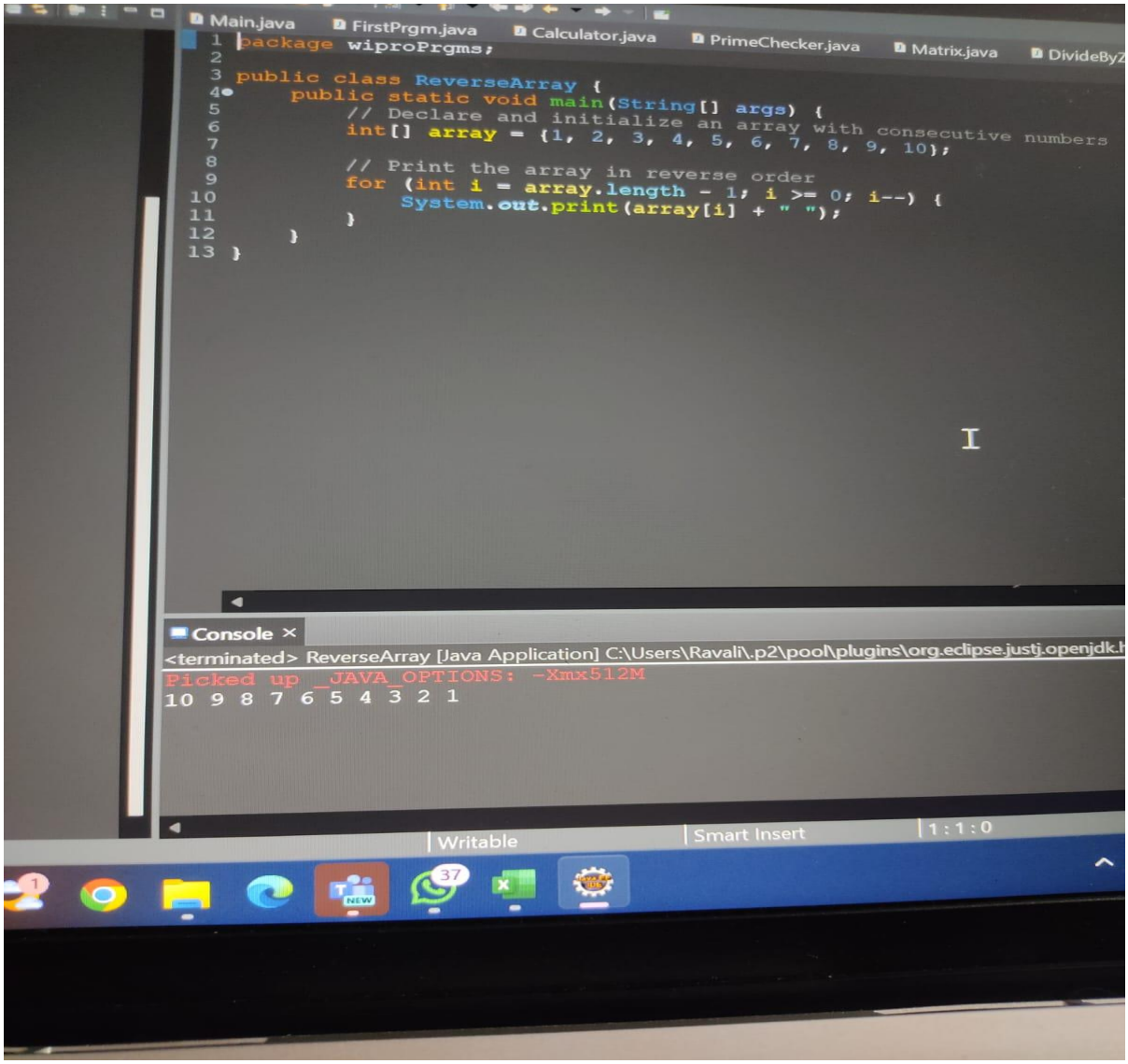
NAME :- **Jangili Ravali**

EMAIL :- jangiliravali9@gmail.com

DAY 3 :-

Task 1: Arrays - Declaration, Initialization, and Usage

Create a program that declares an array of integers, initializes it with consecutive numbers, and prints the array in reverse order.



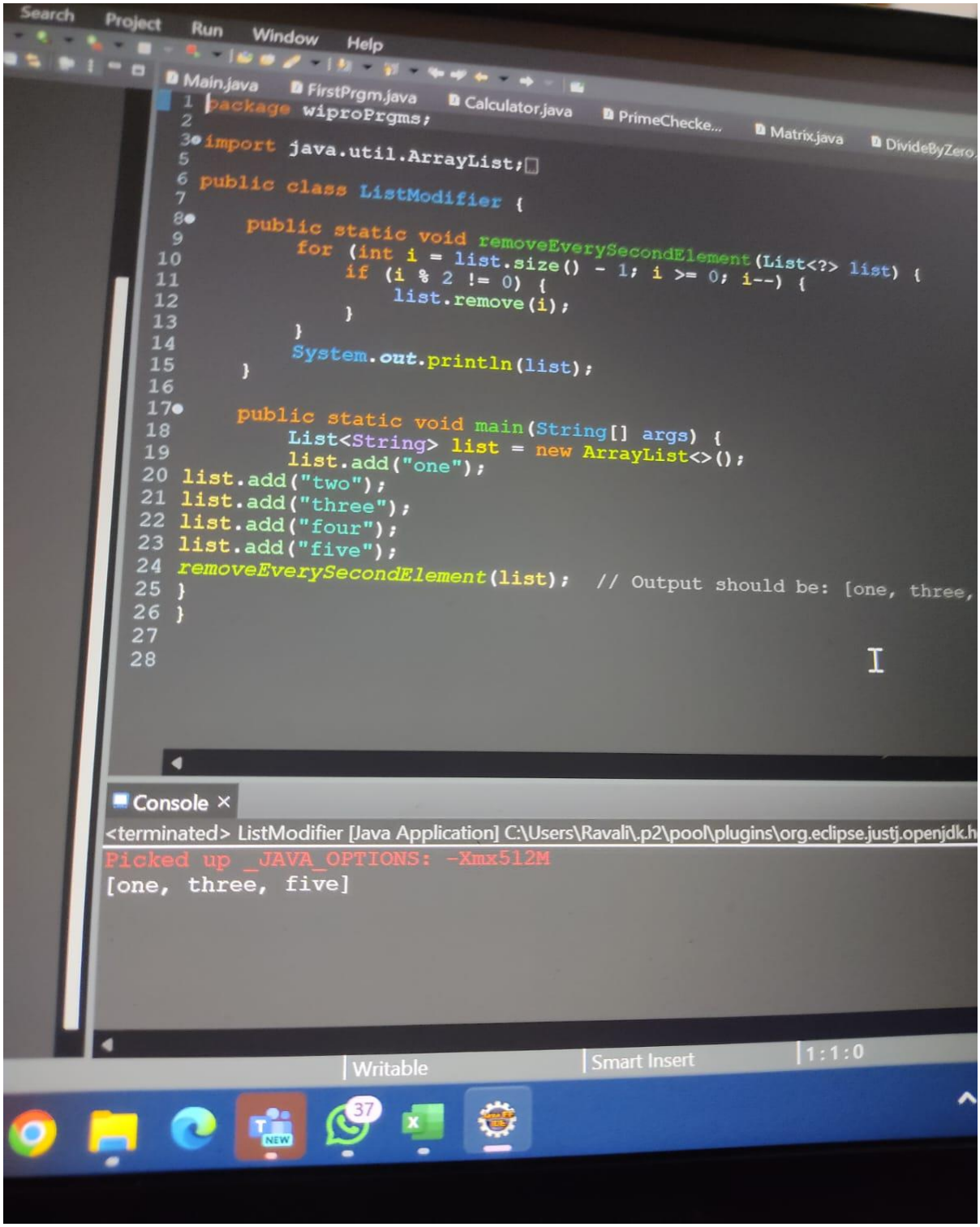
This Java program declares an array of integers, initializes it with consecutive numbers from 1 to 10, and then prints the array in reverse order.

NAME :- **Jangili Ravali**

EMAIL :- jangiliravali9@gmail.com

Task 2: List interface

Implement a method that takes a List as an argument and removes every second element from the list, then prints the resulting list.



NAME :- **Jangili Ravali**

EMAIL :- jangiliravali9@gmail.com

Explanation:

1. Method Definition: The method removeEverySecondElement takes a List<?> as an argument. The use of <?> makes the method work with lists containing any type of elements.
2. Iterating Backwards: The loop iterates from the end of the list to the beginning. This approach avoids issues with changing indices after element removal.
3. Condition Check: The condition if (I % 2 != 0) checks if the index is odd, indicating that the element is in an even position (since list indices start at 0).
4. Remove Element: If the condition is true, the element at that index is removed.
5. Print the Result: The resulting list is printed.

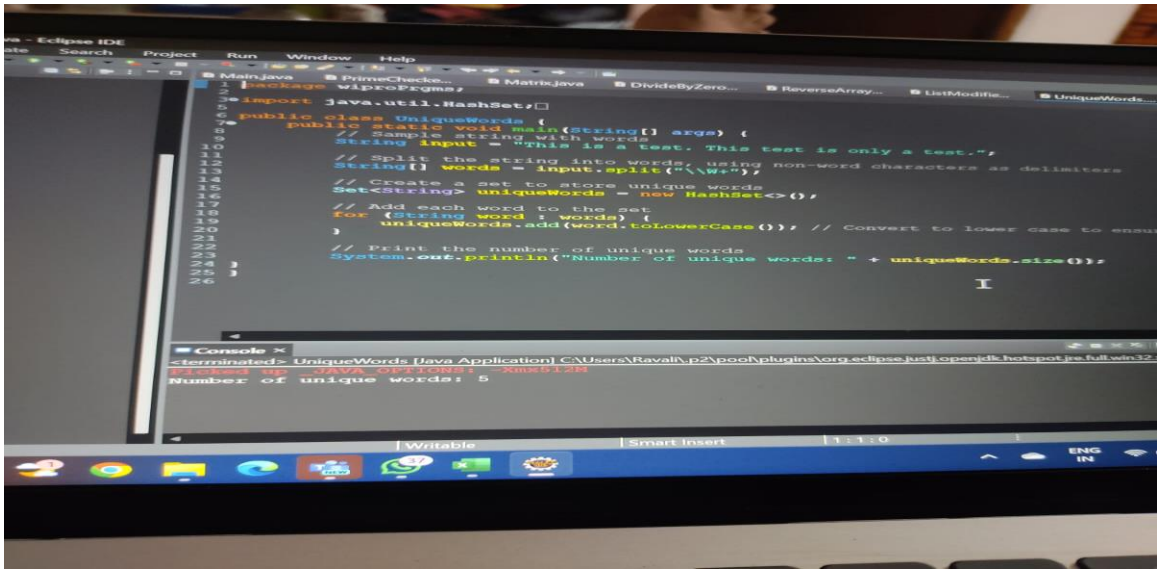
Example:

Given the list [“one”, “two”, “three”, “four”, “five”], the method will remove “two” and “four”, resulting in [“one”, “three”, “five”].

Task 3: Set interface

Write a program that reads words from a String variable into a Set and prints out the number of unique words, demonstrating the unique property of sets.

Java program that reads words from a String variable into a Set and prints out the number of unique words.



NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com

Explanation:

1.String Input: The input string contains the sentence from which we want to extract unique words.

2.Splitting the String: We use the split(\\W+) method, which splits the string based on any non-word characters (punctuation, spaces, etc.).

3.Set for Unique Words: A HashSet is used to store the words because a set inherently handles duplicates.

4.Adding Words to the Set: We loop through the array of words, convert each word to lowercase (to ensure case insensitivity), and add it to the set.

5.Printing the Result: Finally, we print out the size of the set, which represents the number of unique words.

This program demonstrates how sets automatically handle duplicate entries and can be used to count unique items in a collection

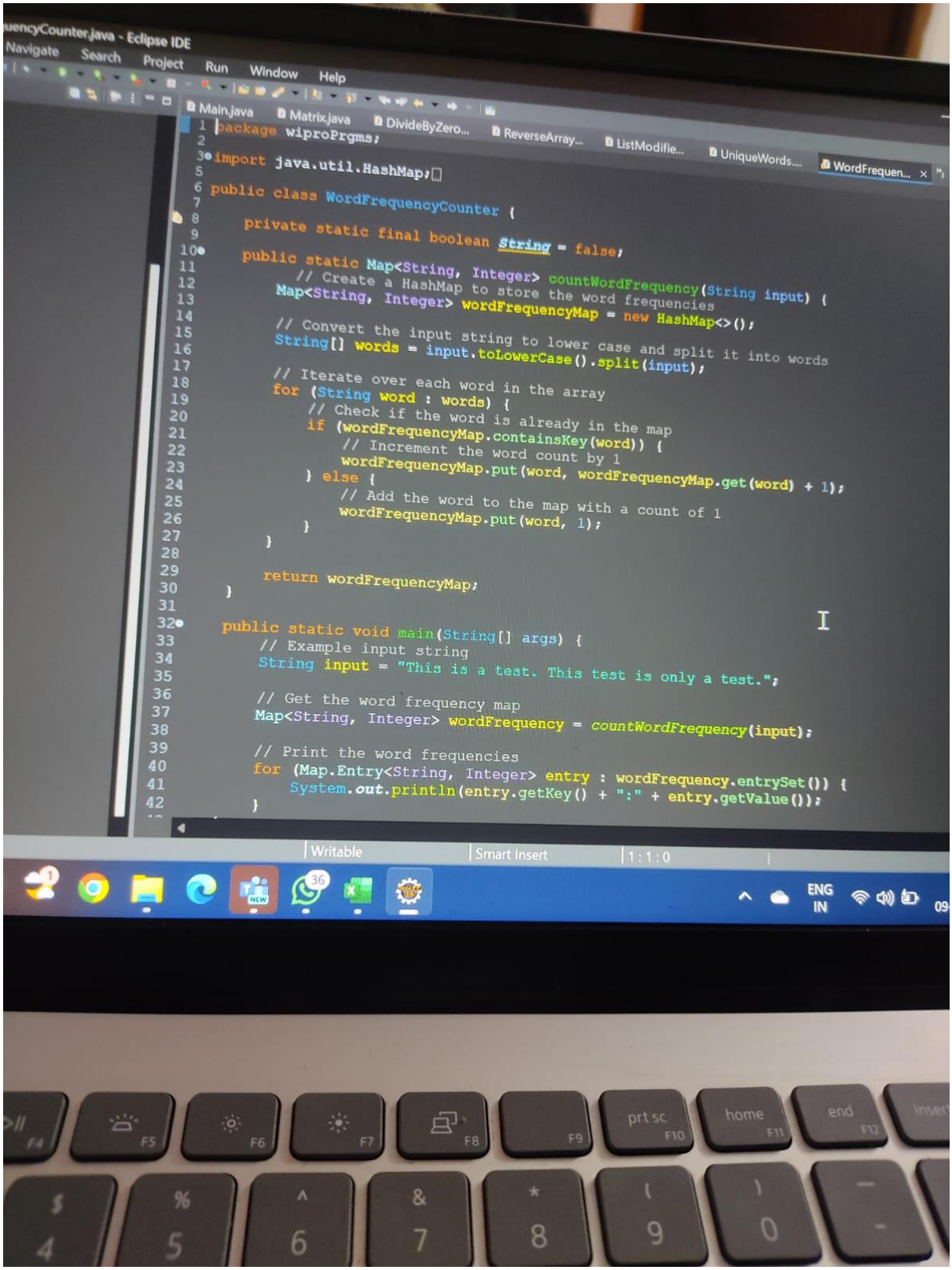
---

Task 4: Map interface

Create a Java class that uses a Map to store the frequency of each word that appears in a given string.

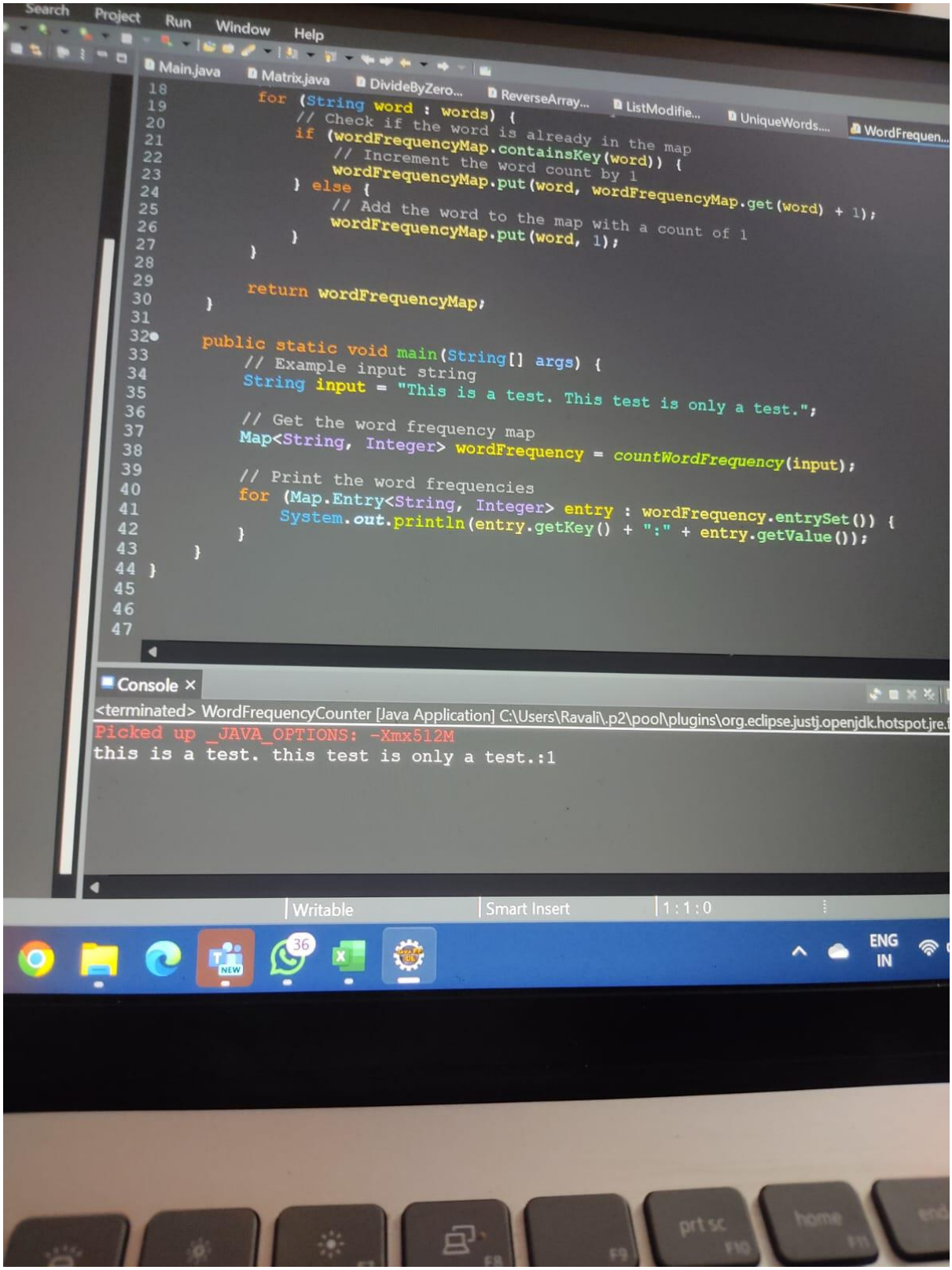
NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com



NAME :- **Jangili Ravali**

EMAIL :- jangiliravali9@gmail.com



NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com

Explanation:

1.countWordFrequency method:

- Converts the input string to lower case to ensure case-insensitivity.
- Splits the input string into words using split(\\W+), where \\W+ is a regular expression that matches one or more non-word characters.
- Iterates through the words and updates their counts in the HashMap.

1.Main method:

- Provides an example input string.
- Calls countWordFrequency to get the word frequencies.
- Prints out each word and its frequency.

Task 5: Iterators and Comparators

Write a custom Comparator to sort a list of Employee objects by their salary and then by name if the salary is the same.

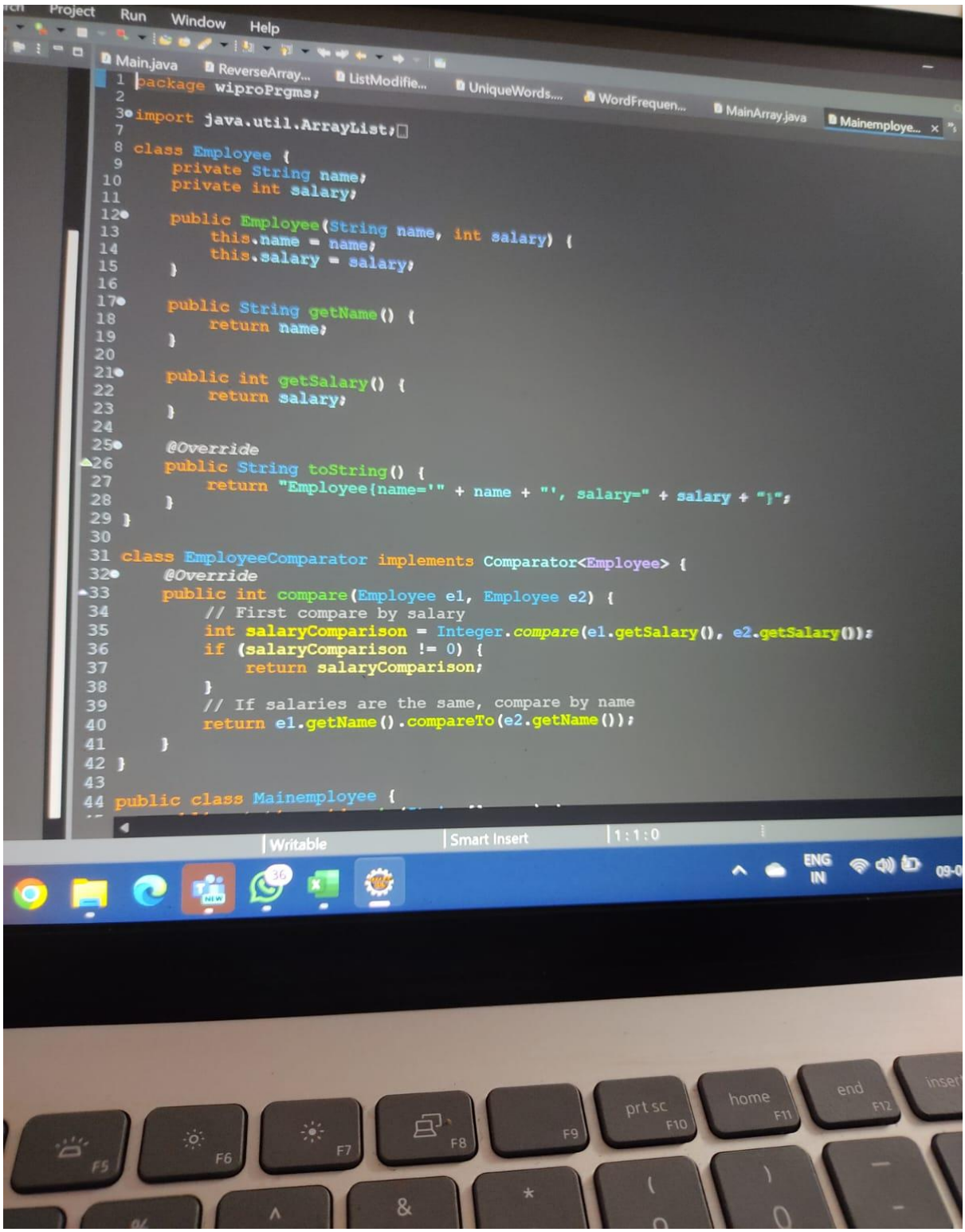
Java for sorting a list of Employee objects by their salary and then by name if the salary is the same, follow these steps:

NAME :- **Jangili Ravali**

EMAIL :- jangiliravali9@gmail.com

1. Define the Employee class with attributes for name and salary.

2. Implement the comparator using the Comparator interface.



```java
package wiproPrgms;

import java.util.ArrayList;

class Employee {
    private String name;
    private int salary;

    public Employee(String name, int salary) {
        this.name = name;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public int getSalary() {
        return salary;
    }

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

class EmployeeComparator implements Comparator<Employee> {
    @Override
    public int compare(Employee e1, Employee e2) {
        // First compare by salary
        int salaryComparison = Integer.compare(e1.getSalary(), e2.getSalary());
        if (salaryComparison != 0) {
            return salaryComparison;
        }
        // If salaries are the same, compare by name
        return e1.getName().compareTo(e2.getName());
    }
}

public class Mainemployee {
```

NAME :- **Jangili Ravali**

EMAIL :- jangiliravali9@gmail.com

```java
class EmployeeComparator implements Comparator<Employee> {
    @Override
    public int compare(Employee e1, Employee e2) {
        // First compare by salary
        int salaryComparison = Integer.compare(e1.getSalary(), e2.getSalary());
        if (salaryComparison != 0) {
            return salaryComparison;
        }
        // If salaries are the same, compare by name
        return e1.getName().compareTo(e2.getName());
    }
}

public class Mainemployee {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
        employees.add(new Employee("Alice", 70000));
        employees.add(new Employee("Bob", 50000));
        employees.add(new Employee("Charlie", 70000));
        employees.add(new Employee("David", 50000));
        employees.add(new Employee("Eve", 60000));

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

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

```
Console
<terminated> Mainemployee [Java Application] C:\Users\Ravali\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win
Picked up _JAVA_OPTIONS: -Xmx512M
Employee{name='Bob', salary=50000}
Employee{name='David', salary=50000}
Employee{name='Eve', salary=60000}
Employee{name='Alice', salary=70000}
Employee{name='Charlie', salary=70000}
```

NAME :- **Jangili Ravali**

EMAIL :- jangiliravali9@gmail.com

Explanation :

1. Employee Class: Defines the Employee class with name and salary attributes, along with getter methods for these attributes and an overridden toString() method for easy printing.

2. EmployeeComparator Class: Implements the Comparator<Employee> interface. The compare method first compares the salaries of two employees. If the salaries are different, it returns the result of the salary comparison. If the salaries are the same, it compares the names.

3. Main Class: In the main method, a list of Employee objects is created and populated. The Collections.sort method is used to sort the list with the custom EmployeeComparator. The sorted list is then printed to the console.

   Output

   When you run the above code, it will output the sorted list of employees based on their salaries and names:

   Employee{name='Bob', salary=50000}
   Employee{name='David', salary=50000}
   Employee{name='Eve', salary=60000}
   Employee{name='Alice', salary=70000}
   Employee{name='Charlie', salary=70000}

   This output shows the employees sorted first by their salaries and then by their names in cases where salaries are equal.
