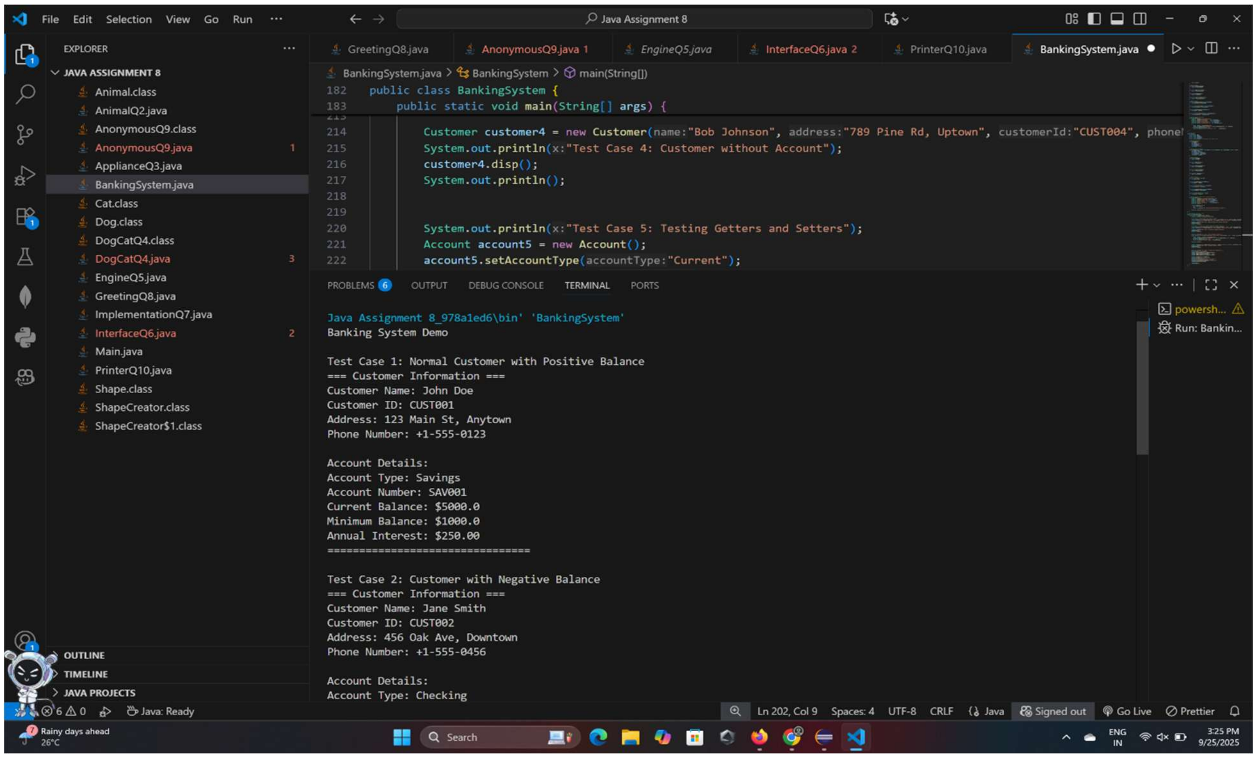
Assignment -11 (24-09-2025)

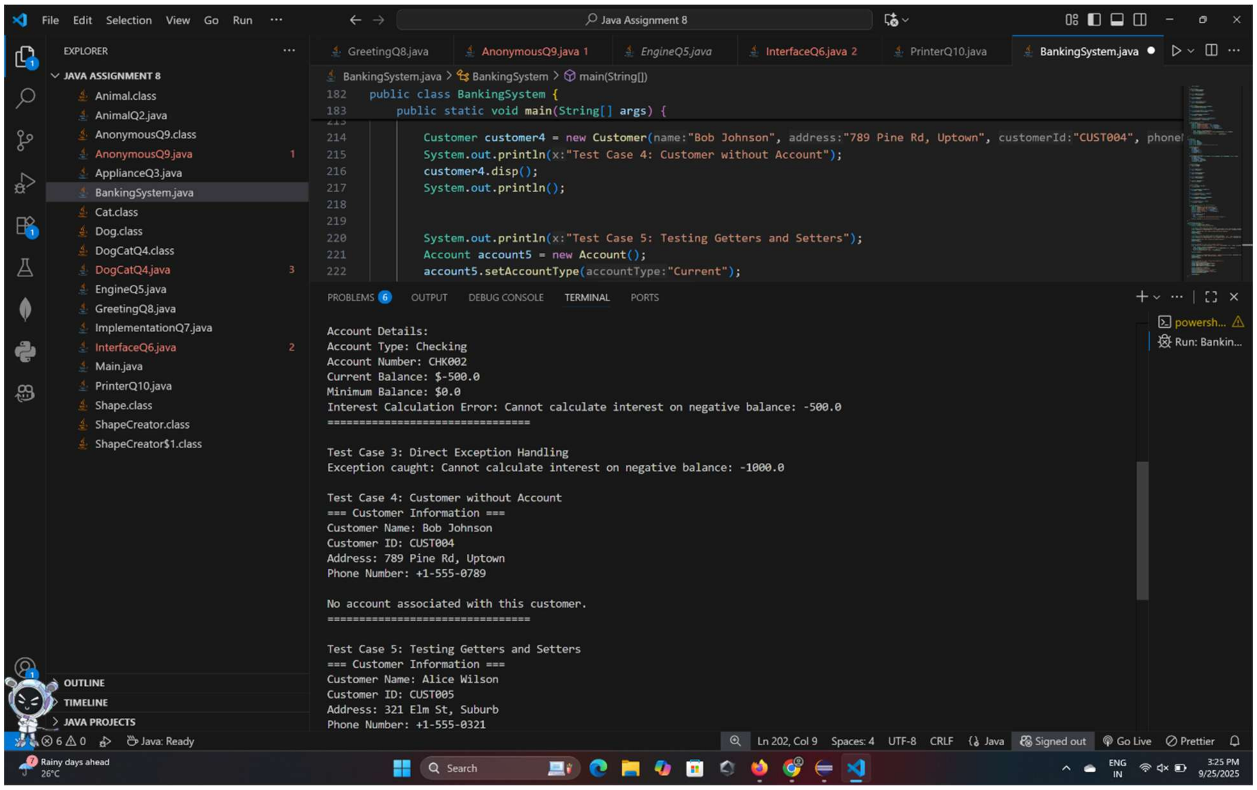
Roll No - 107

Problem 1: Create Customer class with the relevant information like name, address, id, phone



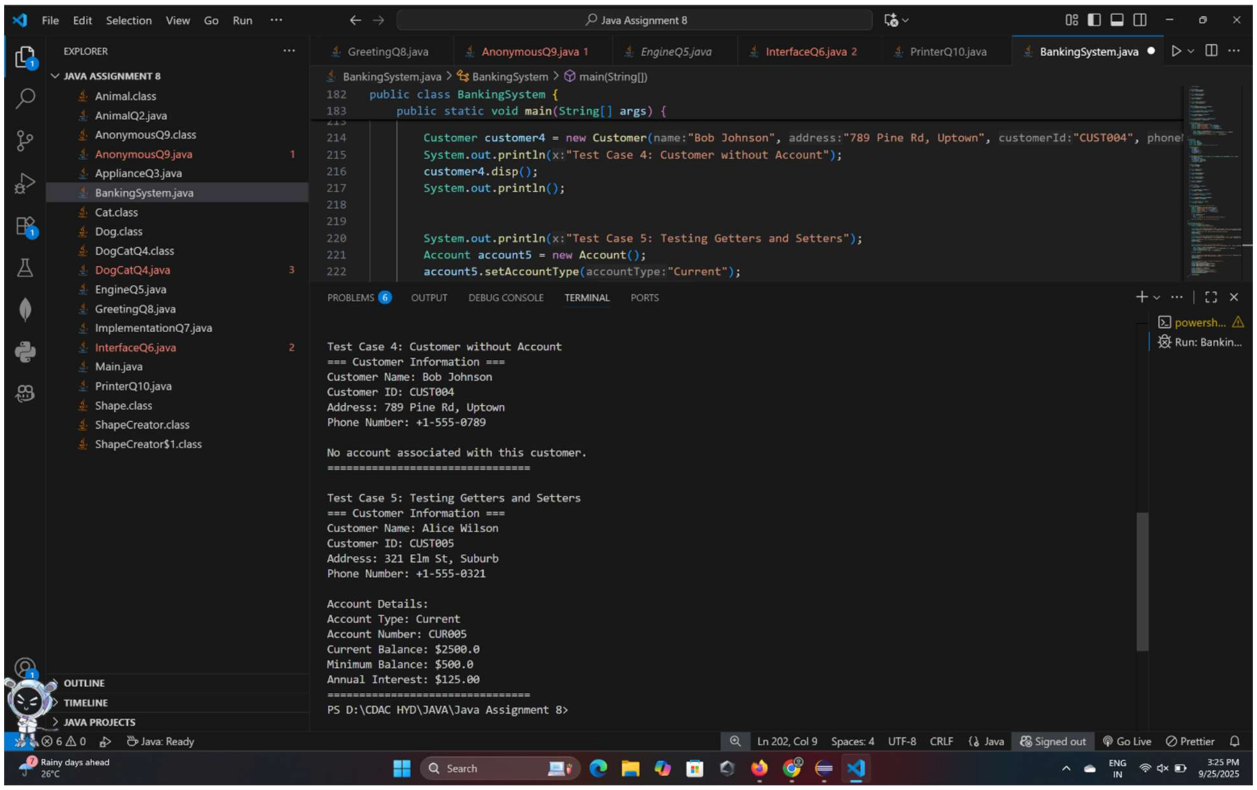
```java
public class BankingSystem {
    public static void main(String[] args) {
        Customer customer4 = new Customer(name:"Bob Johnson", address:"789 Pine Rd, Uptown", customerId:"CUST004", phone
        System.out.println(x:"Test Case 4: Customer without Account");
        customer4.disp();
        System.out.println();


        System.out.println(x:"Test Case 5: Testing Getters and Setters");
        Account account5 = new Account();
        account5.setAccountType(accountType:"Current");
```

```
Java Assignment 8_978a1ed6\bin' 'BankingSystem'
Banking System Demo

Test Case 1: Normal Customer with Positive Balance
=== Customer Information ===
Customer Name: John Doe
Customer ID: CUST001
Address: 123 Main St, Anytown
Phone Number: +1-555-0123

Account Details:
Account Type: Savings
Account Number: SAV001
Current Balance: $5000.0
Minimum Balance: $1000.0
Annual Interest: $250.00
================================

Test Case 2: Customer with Negative Balance
=== Customer Information ===
Customer Name: Jane Smith
Customer ID: CUST002
Address: 456 Oak Ave, Downtown
Phone Number: +1-555-0456

Account Details:
Account Type: Checking
```



```java
public class BankingSystem {
    public static void main(String[] args) {
        Customer customer4 = new Customer(name:"Bob Johnson", address:"789 Pine Rd, Uptown", customerId:"CUST004", phone
        System.out.println(x:"Test Case 4: Customer without Account");
        customer4.disp();
        System.out.println();


        System.out.println(x:"Test Case 5: Testing Getters and Setters");
        Account account5 = new Account();
        account5.setAccountType(accountType:"Current");
```

```
Account Details:
Account Type: Checking
Account Number: CHK002
Current Balance: $-500.0
Minimum Balance: $0.0
Interest Calculation Error: Cannot calculate interest on negative balance: -500.0
===============================

Test Case 3: Direct Exception Handling
Exception caught: Cannot calculate interest on negative balance: -1000.0

Test Case 4: Customer without Account
=== Customer Information ===
Customer Name: Bob Johnson
Customer ID: CUST004
Address: 789 Pine Rd, Uptown
Phone Number: +1-555-0789

No account associated with this customer.
===============================

Test Case 5: Testing Getters and Setters
=== Customer Information ===
Customer Name: Alice Wilson
Customer ID: CUST005
Address: 321 Elm St, Suburb
Phone Number: +1-555-0321
```

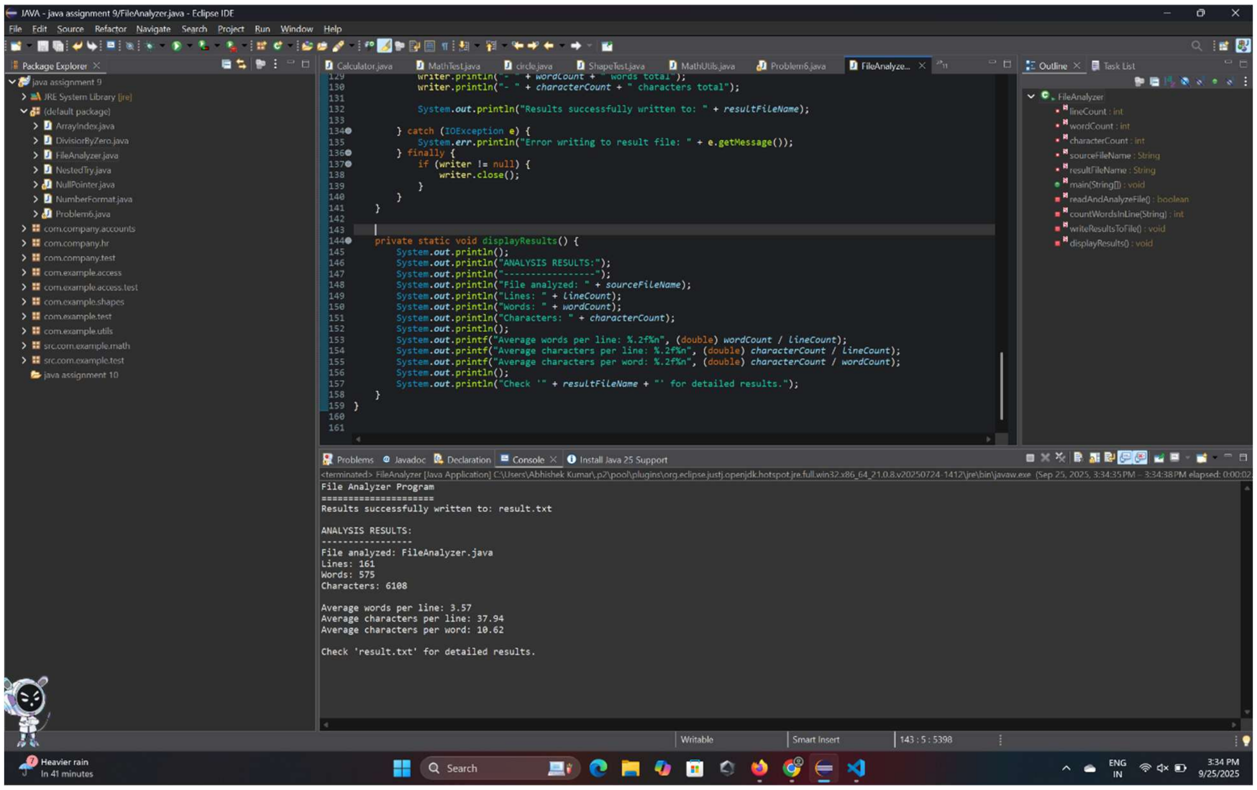


```java
public class BankingSystem {
    public static void main(String[] args) {
        Customer customer4 = new Customer(name:"Bob Johnson", address:"789 Pine Rd, Uptown", customerId:"CUST004", phone
        System.out.println(x:"Test Case 4: Customer without Account");
        customer4.disp();
        System.out.println();


        System.out.println(x:"Test Case 5: Testing Getters and Setters");
        Account account5 = new Account();
        account5.setAccountType(accountType:"Current");
```

```
Test Case 4: Customer without Account
=== Customer Information ===
Customer Name: Bob Johnson
Customer ID: CUST004
Address: 789 Pine Rd, Uptown
Phone Number: +1-555-0789

No account associated with this customer.
===============================

Test Case 5: Testing Getters and Setters
=== Customer Information ===
Customer Name: Alice Wilson
Customer ID: CUST005
Address: 321 Elm St, Suburb
Phone Number: +1-555-0321

Account Details:
Account Type: Current
Account Number: CUR005
Current Balance: $2500.0
Minimum Balance: $500.0
Annual Interest: $125.00
===============================
PS D:\CDAC HYD\JAVA\Java Assignment 8>
```

Java IO:

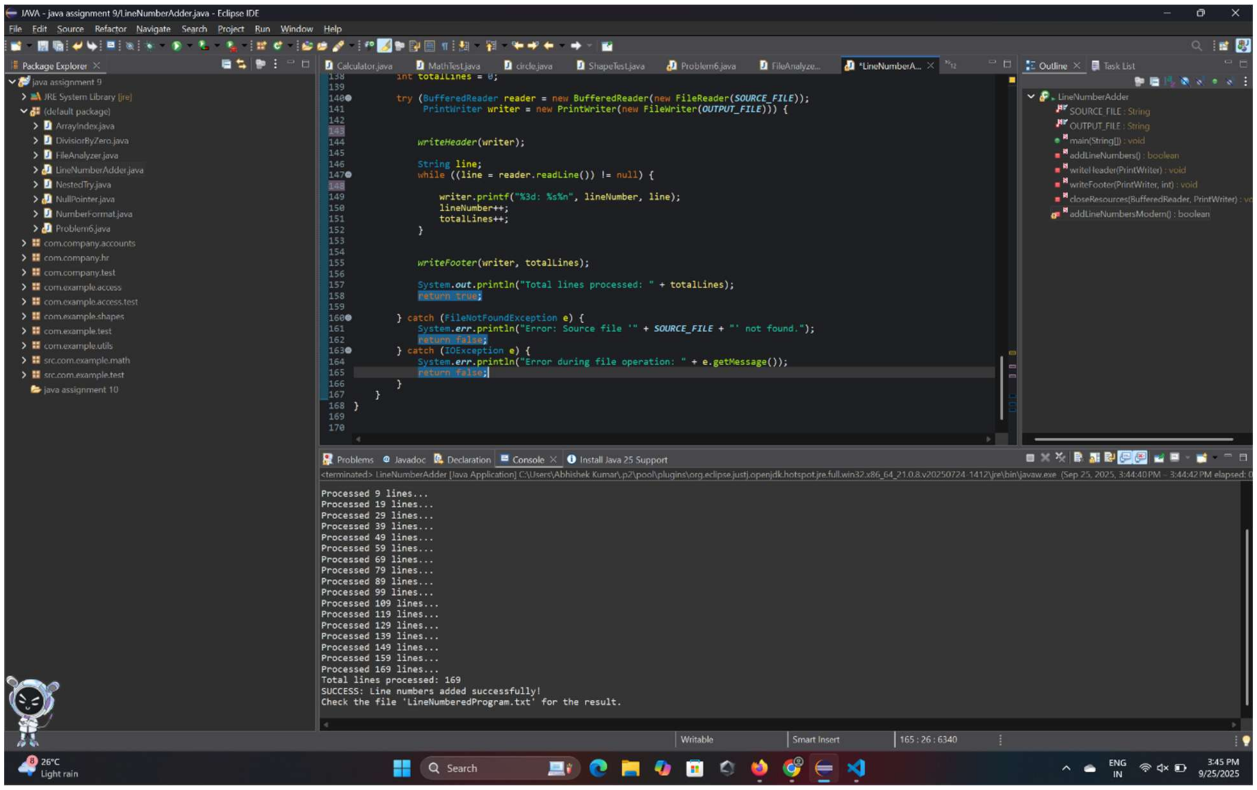
1. Write a Program to read the same program file and find the no. of lines, words and characters. Write the result in in to a text file (result.txt)



```java
            writer.println("- " + wordCount + " words total");
            writer.println("- " + characterCount + " characters total");

            System.out.println("Results successfully written to: " + resultFileName);

        } catch (IOException e) {
            System.err.println("Error writing to result file: " + e.getMessage());
        } finally {
            if (writer != null) {
                writer.close();
            }
        }
    }

    
    private static void displayResults() {
        System.out.println();
        System.out.println("ANALYSIS RESULTS:");
        System.out.println("-----------------");
        System.out.println("File analyzed: " + sourceFileName);
        System.out.println("Lines: " + lineCount);
        System.out.println("Words: " + wordCount);
        System.out.println("Characters: " + characterCount);
        System.out.println();
        System.out.printf("Average words per line: %.2f%n", (double) wordCount / lineCount);
        System.out.printf("Average characters per line: %.2f%n", (double) characterCount / lineCount);
        System.out.printf("Average characters per word: %.2f%n", (double) characterCount / wordCount);
        System.out.println();
        System.out.println("Check '" + resultFileName + "' for detailed results.");
    }
}
```

```
<terminated> FileAnalyzer [Java Application] C:\Users\Abhishek Kumar\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.8.v20250724-1412\jre\bin\javaw.exe (Sep 25, 2025, 3:34:35 PM – 3:34:38 PM elapsed: 0:00:02
File Analyzer Program
=====================
Results successfully written to: result.txt

ANALYSIS RESULTS:
-----------------
File analyzed: FileAnalyzer.java
Lines: 161
Words: 575
Characters: 6108

Average words per line: 3.57
Average characters per line: 37.94
Average characters per word: 10.62

Check 'result.txt' for detailed results.
```

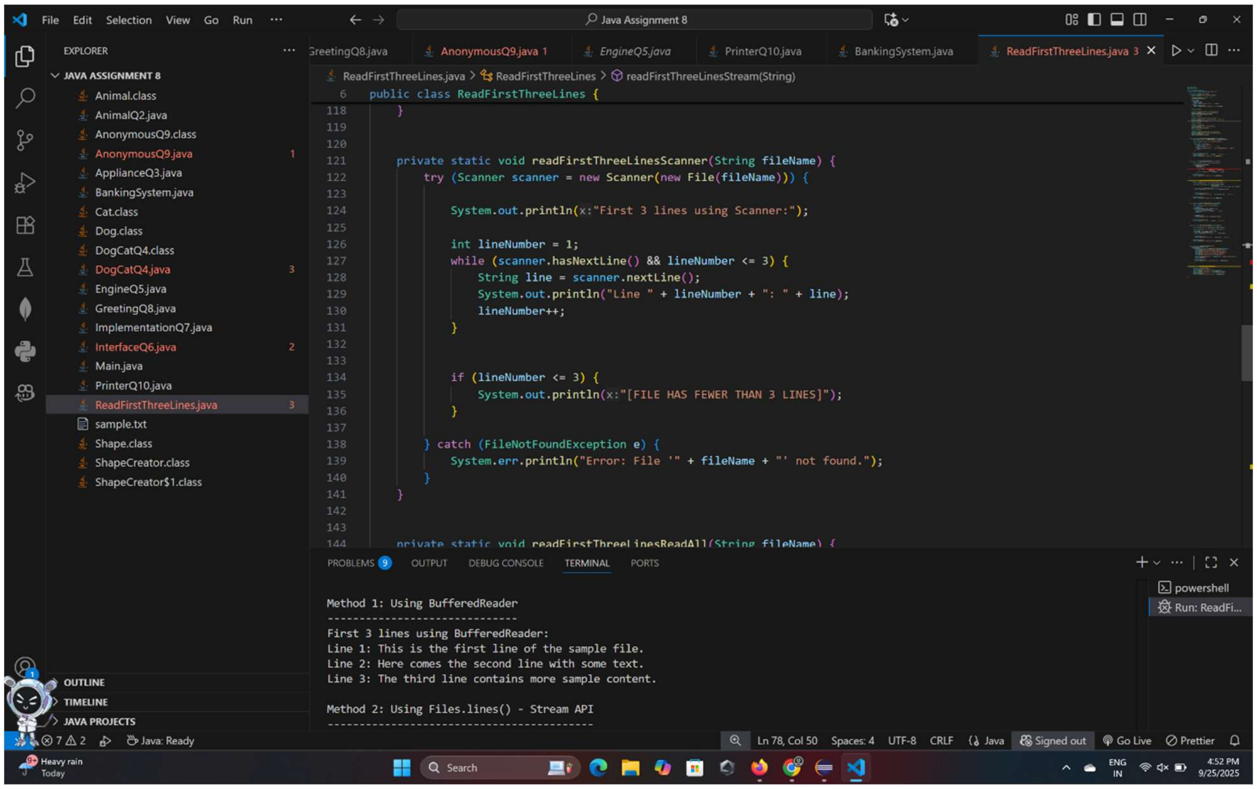
2. Write a program to read the same program file and write it to other file with the lines number added before each line, starting from 1.



```java
        int totalLines = 0;

        try (BufferedReader reader = new BufferedReader(new FileReader(SOURCE_FILE));
             PrintWriter writer = new PrintWriter(new FileWriter(OUTPUT_FILE))) {


            writeHeader(writer);

            String line;
            while ((line = reader.readLine()) != null) {

                writer.printf("%3d: %s%n", lineNumber, line);
                lineNumber++;
                totalLines++;
            }


            writeFooter(writer, totalLines);

            System.out.println("Total lines processed: " + totalLines);
            return true;

        } catch (FileNotFoundException e) {
            System.err.println("Error: Source file '" + SOURCE_FILE + "' not found.");
            return false;
        } catch (IOException e) {
            System.err.println("Error during file operation: " + e.getMessage());
            return false;
        }
    }
}
```

```
Processed 9 lines...
Processed 19 lines...
Processed 29 lines...
Processed 39 lines...
Processed 49 lines...
Processed 59 lines...
Processed 69 lines...
Processed 79 lines...
Processed 89 lines...
Processed 99 lines...
Processed 109 lines...
Processed 119 lines...
Processed 129 lines...
Processed 139 lines...
Processed 149 lines...
Processed 159 lines...
Processed 169 lines...
Total lines processed: 169
SUCCESS: Line numbers added successfully!
Check the file 'LineNumberedProgram.txt' for the result.
```

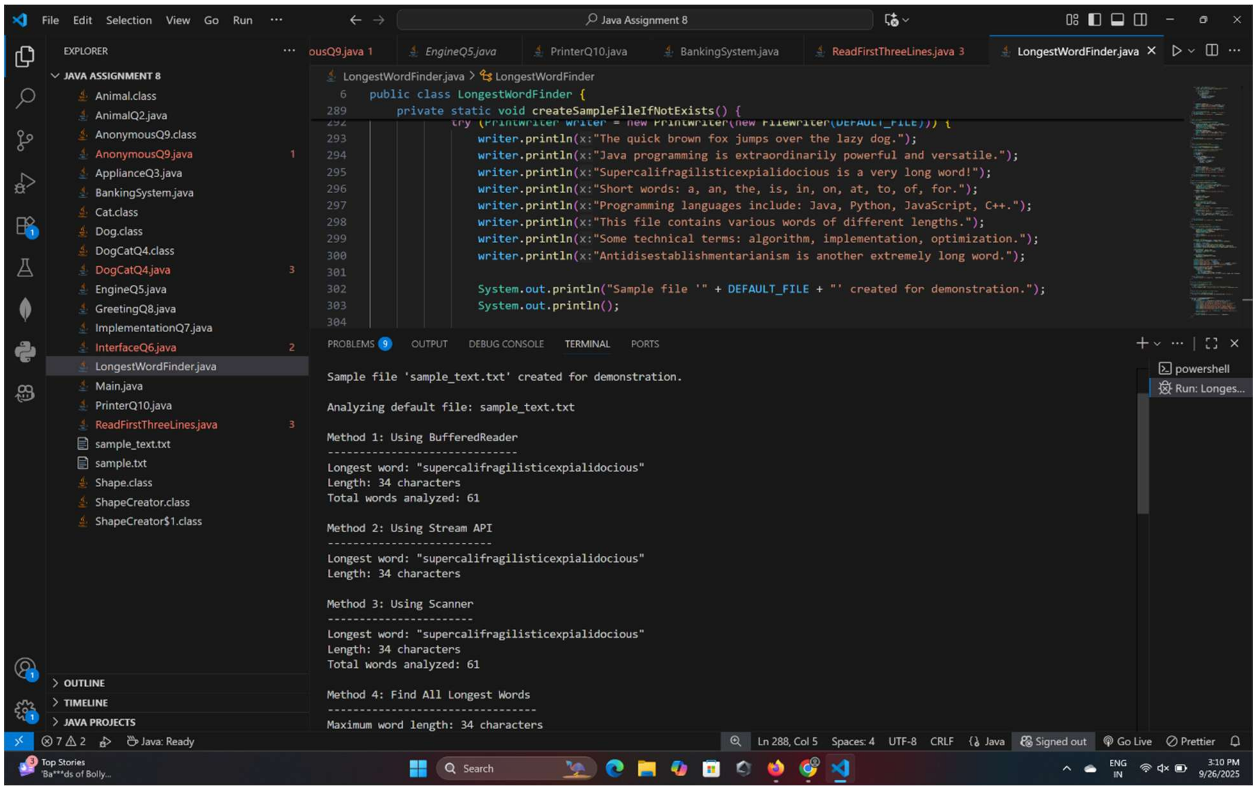
3. Write a Java program to read first 3 lines from a file.



```java
public class ReadFirstThreeLines {
    }


    private static void readFirstThreeLinesScanner(String fileName) {
        try (Scanner scanner = new Scanner(new File(fileName))) {

            System.out.println(x:"First 3 lines using Scanner:");

            int lineNumber = 1;
            while (scanner.hasNextLine() && lineNumber <= 3) {
                String line = scanner.nextLine();
                System.out.println("Line " + lineNumber + ": " + line);
                lineNumber++;
            }


            if (lineNumber <= 3) {
                System.out.println(x:"[FILE HAS FEWER THAN 3 LINES]");
            }

        } catch (FileNotFoundException e) {
            System.err.println("Error: File '" + fileName + "' not found.");
        }
    }


    private static void readFirstThreeLinesReadAll(String fileName) {
```

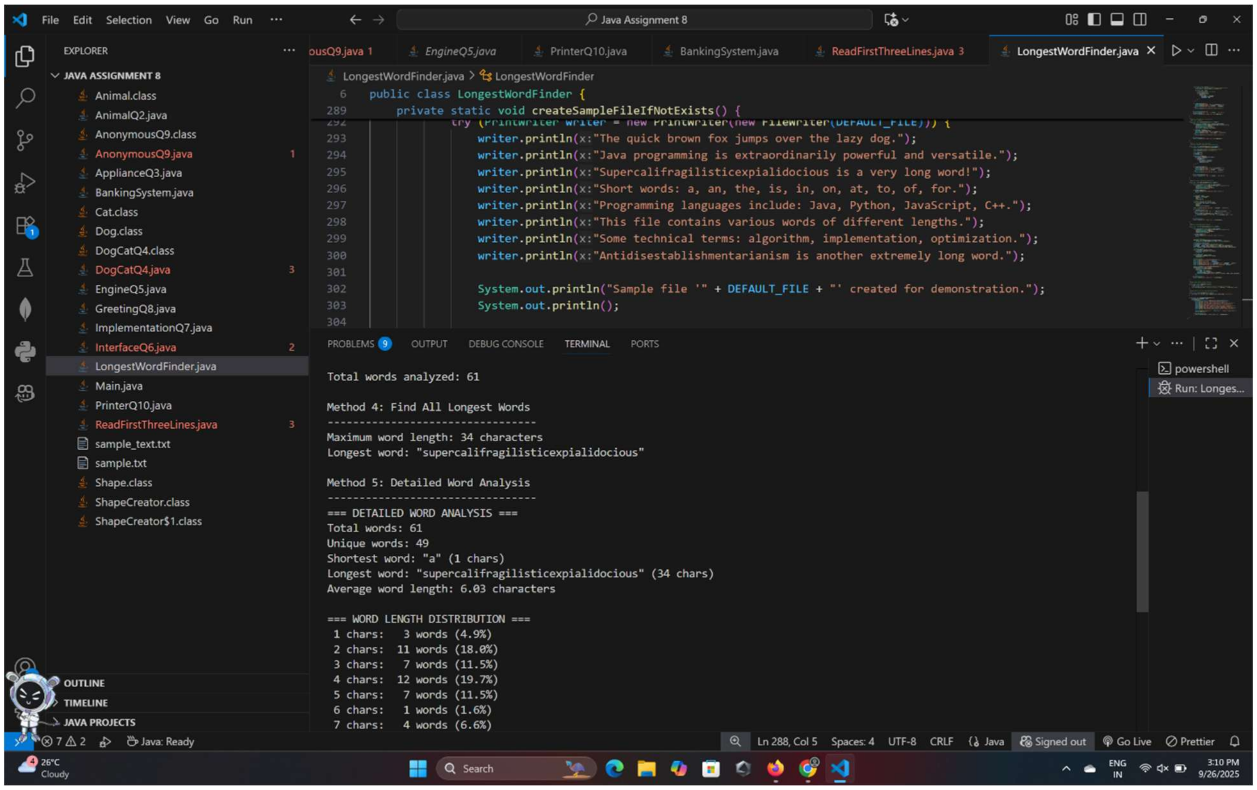
```
Method 1: Using BufferedReader
-----------------------------
First 3 lines using BufferedReader:
Line 1: This is the first line of the sample file.
Line 2: Here comes the second line with some text.
Line 3: The third line contains more sample content.

Method 2: Using Files.lines() - Stream API
------------------------------------------
```

4. Write a Java program to find the longest word in a text file



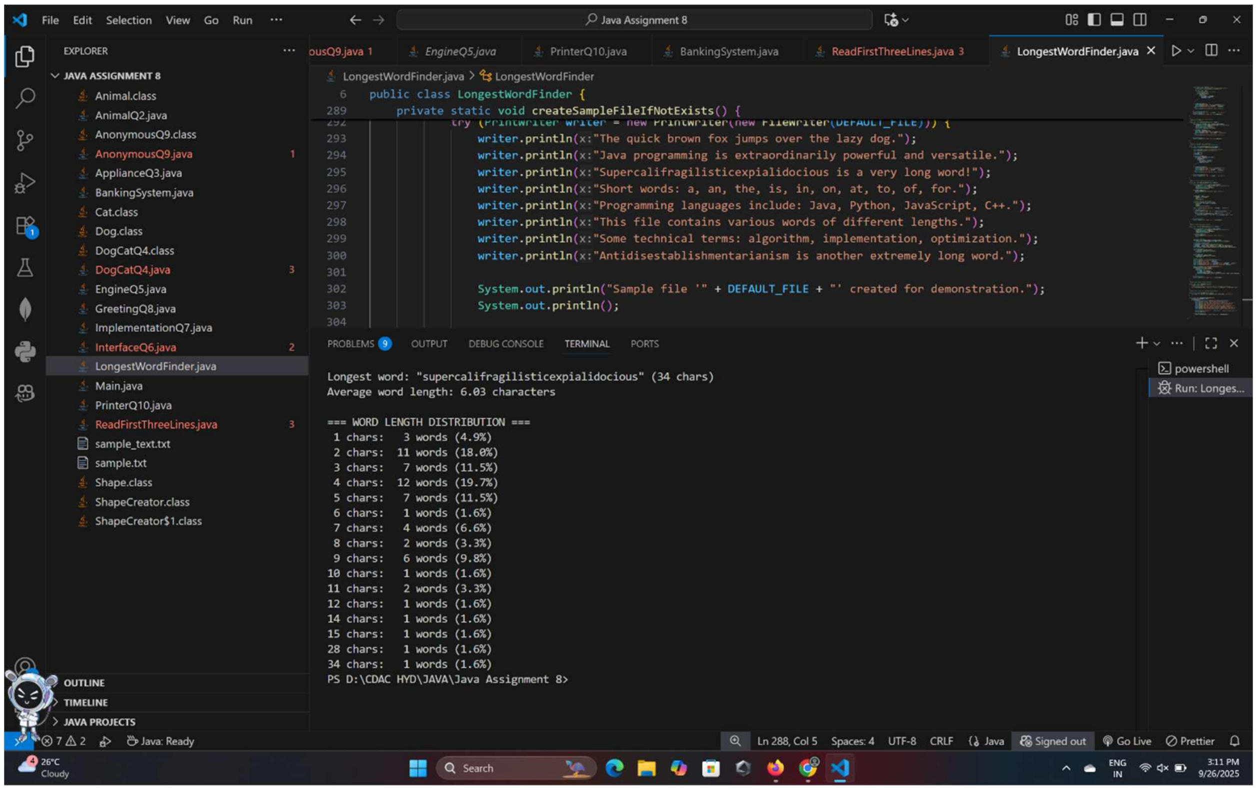
```java
public class LongestWordFinder {
    private static void createSampleFileIfNotExists() {
        try (PrintWriter writer = new PrintWriter(new FileWriter(DEFAULT_FILE))) {
            writer.println(x:"The quick brown fox jumps over the lazy dog.");
            writer.println(x:"Java programming is extraordinarily powerful and versatile.");
            writer.println(x:"Supercalifragilisticexpialidocious is a very long word!");
            writer.println(x:"Short words: a, an, the, is, in, on, at, to, of, for.");
            writer.println(x:"Programming languages include: Java, Python, JavaScript, C++.");
            writer.println(x:"This file contains various words of different lengths.");
            writer.println(x:"Some technical terms: algorithm, implementation, optimization.");
            writer.println(x:"Antidisestablishmentarianism is another extremely long word.");

            System.out.println("Sample file '" + DEFAULT_FILE + "' created for demonstration.");
            System.out.println();
```

```
Sample file 'sample_text.txt' created for demonstration.

Analyzing default file: sample_text.txt

Method 1: Using BufferedReader
------------------------------
Longest word: "supercalifragilisticexpialidocious"
Length: 34 characters
Total words analyzed: 61

Method 2: Using Stream API
--------------------------
Longest word: "supercalifragilisticexpialidocious"
Length: 34 characters

Method 3: Using Scanner
-----------------------
Longest word: "supercalifragilisticexpialidocious"
Length: 34 characters
Total words analyzed: 61

Method 4: Find All Longest Words
--------------------------------
Maximum word length: 34 characters
```



```java
public class LongestWordFinder {
    private static void createSampleFileIfNotExists() {
        try (PrintWriter writer = new PrintWriter(new FileWriter(DEFAULT_FILE))) {
            writer.println(x:"The quick brown fox jumps over the lazy dog.");
            writer.println(x:"Java programming is extraordinarily powerful and versatile.");
            writer.println(x:"Supercalifragilisticexpialidocious is a very long word!");
            writer.println(x:"Short words: a, an, the, is, in, on, at, to, of, for.");
            writer.println(x:"Programming languages include: Java, Python, JavaScript, C++.");
            writer.println(x:"This file contains various words of different lengths.");
            writer.println(x:"Some technical terms: algorithm, implementation, optimization.");
            writer.println(x:"Antidisestablishmentarianism is another extremely long word.");

            System.out.println("Sample file '" + DEFAULT_FILE + "' created for demonstration.");
            System.out.println();
```

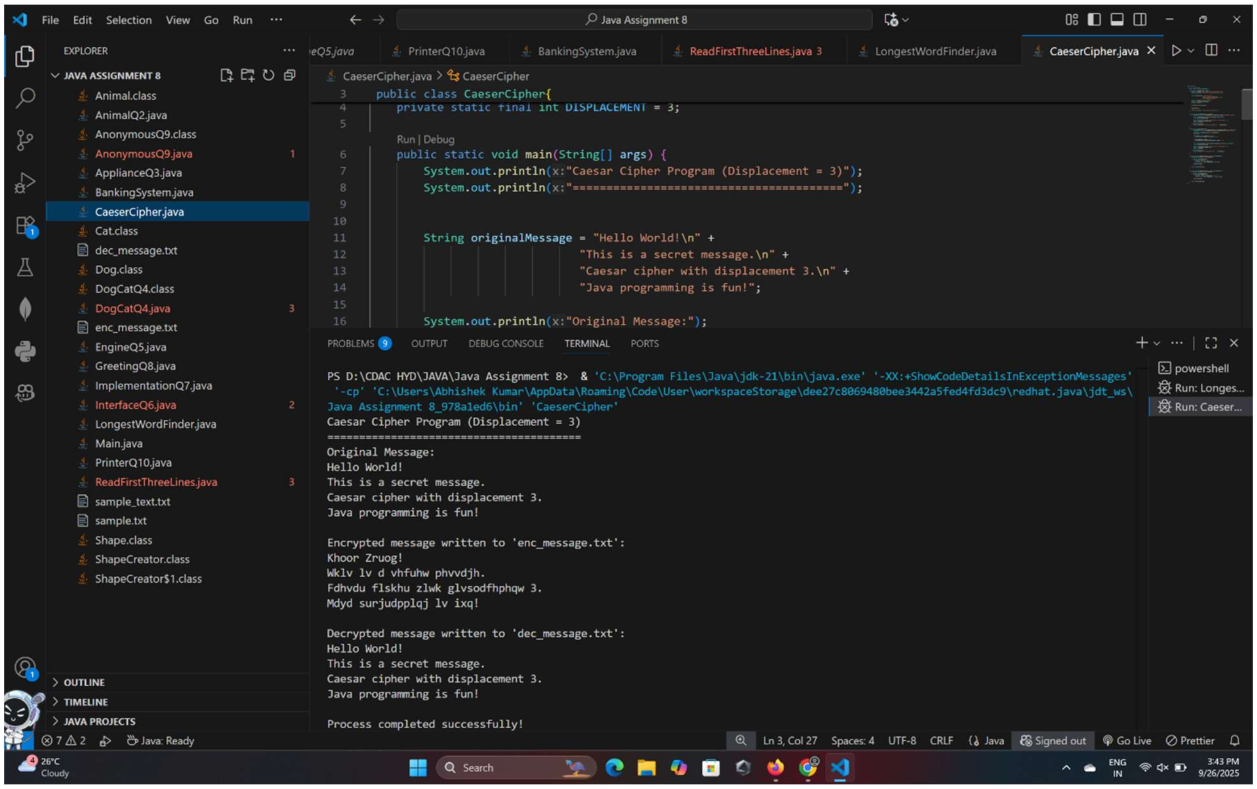
```
Total words analyzed: 61

Method 4: Find All Longest Words
--------------------------------
Maximum word length: 34 characters
Longest word: "supercalifragilisticexpialidocious"

Method 5: Detailed Word Analysis
--------------------------------
=== DETAILED WORD ANALYSIS ===
Total words: 61
Unique words: 49
Shortest word: "a" (1 chars)
Longest word: "supercalifragilisticexpialidocious" (34 chars)
Average word length: 6.03 characters

=== WORD LENGTH DISTRIBUTION ===
 1 chars:   3 words (4.9%)
 2 chars:  11 words (18.0%)
 3 chars:   7 words (11.5%)
 4 chars:  12 words (19.7%)
 5 chars:   7 words (11.5%)
 6 chars:   1 words (1.6%)
 7 chars:   4 words (6.6%)
```



```java
public class LongestWordFinder {
    private static void createSampleFileIfNotExists() {
            try (PrintWriter writer = new PrintWriter(new FileWriter(DEFAULT_FILE))) {
                writer.println(x:"The quick brown fox jumps over the lazy dog.");
                writer.println(x:"Java programming is extraordinarily powerful and versatile.");
                writer.println(x:"Supercalifragilisticexpialidocious is a very long word!");
                writer.println(x:"Short words: a, an, the, is, in, on, at, to, of, for.");
                writer.println(x:"Programming languages include: Java, Python, JavaScript, C++.");
                writer.println(x:"This file contains various words of different lengths.");
                writer.println(x:"Some technical terms: algorithm, implementation, optimization.");
                writer.println(x:"Antidisestablishmentarianism is another extremely long word.");

                System.out.println("Sample file '" + DEFAULT_FILE + "' created for demonstration.");
                System.out.println();
```

```
Longest word: "supercalifragilisticexpialidocious" (34 chars)
Average word length: 6.03 characters

=== WORD LENGTH DISTRIBUTION ===
 1 chars:   3 words (4.9%)
 2 chars:  11 words (18.0%)
 3 chars:   7 words (11.5%)
 4 chars:  12 words (19.7%)
 5 chars:   7 words (11.5%)
 6 chars:   1 words (1.6%)
 7 chars:   4 words (6.6%)
 8 chars:   2 words (3.3%)
 9 chars:   6 words (9.8%)
10 chars:   1 words (1.6%)
11 chars:   2 words (3.3%)
12 chars:   1 words (1.6%)
14 chars:   1 words (1.6%)
15 chars:   1 words (1.6%)
28 chars:   1 words (1.6%)
34 chars:   1 words (1.6%)
PS D:\CDAC HYD\JAVA\Java Assignment 8>
```

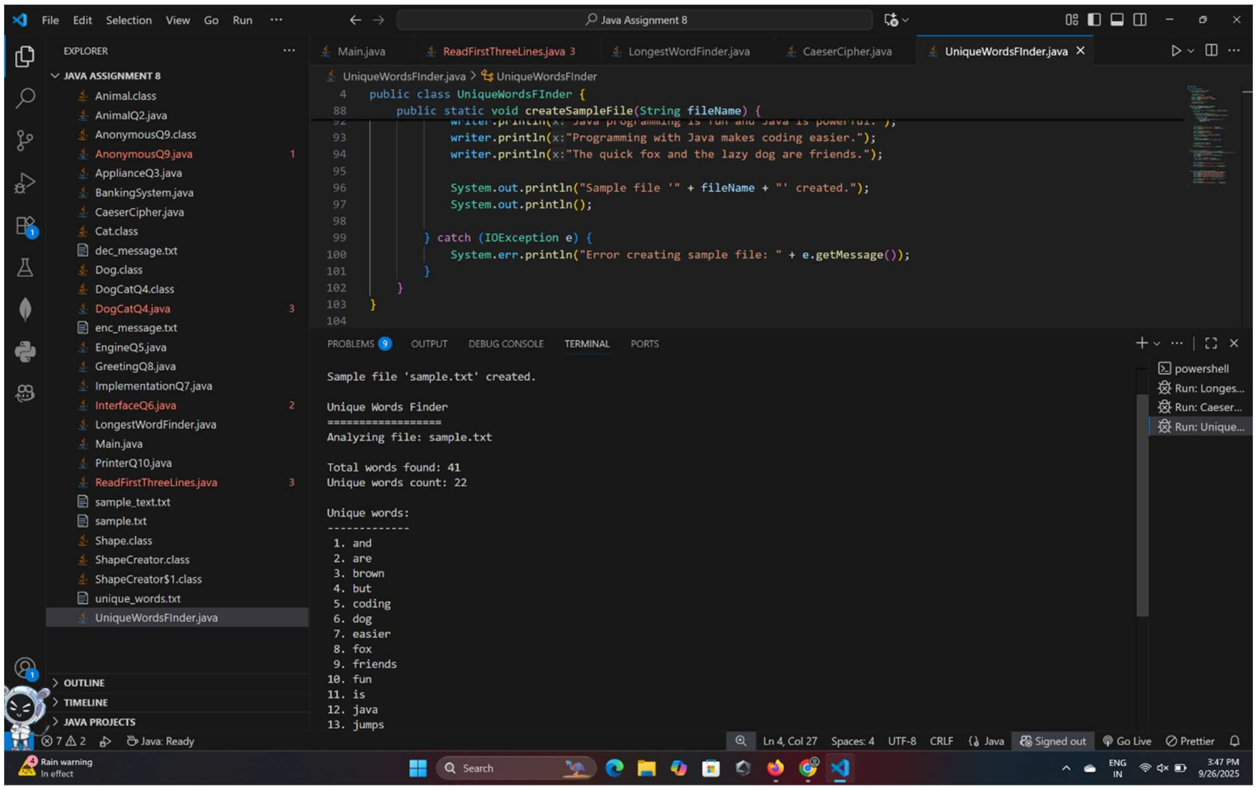
5. Write a programs to implemnt Caeser cipher using files.



```java
public class CaeserCipher{
    private static final int DISPLACEMENT = 3;

    public static void main(String[] args) {
        System.out.println(x:"Caesar Cipher Program (Displacement = 3)");
        System.out.println(x:"========================================");


        String originalMessage = "Hello World!\n" +
                                 "This is a secret message.\n" +
                                 "Caesar cipher with displacement 3.\n" +
                                 "Java programming is fun!";

        System.out.println(x:"Original Message:");
```

```
PS D:\CDAC HYD\JAVA\Java Assignment 8>  & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages'
 '-cp' 'C:\Users\Abhishek Kumar\AppData\Roaming\Code\User\workspaceStorage\dee27c8069480bee3442a5fed4fd3dc9\redhat.java\jdt_ws\
Java Assignment 8_978a1ed6\bin' 'CaeserCipher'
Caesar Cipher Program (Displacement = 3)
========================================
Original Message:
Hello World!
This is a secret message.
Caesar cipher with displacement 3.
Java programming is fun!

Encrypted message written to 'enc_message.txt':
Khoor Zruog!
Wklv lv d vhfuhw phvvdjh.
Fdhvdu flskhu zlwk glvsodfhphqw 3.
Mdyd surjudpplqj lv ixq!

Decrypted message written to 'dec_message.txt':
Hello World!
This is a secret message.
Caesar cipher with displacement 3.
Java programming is fun!

Process completed successfully!
```

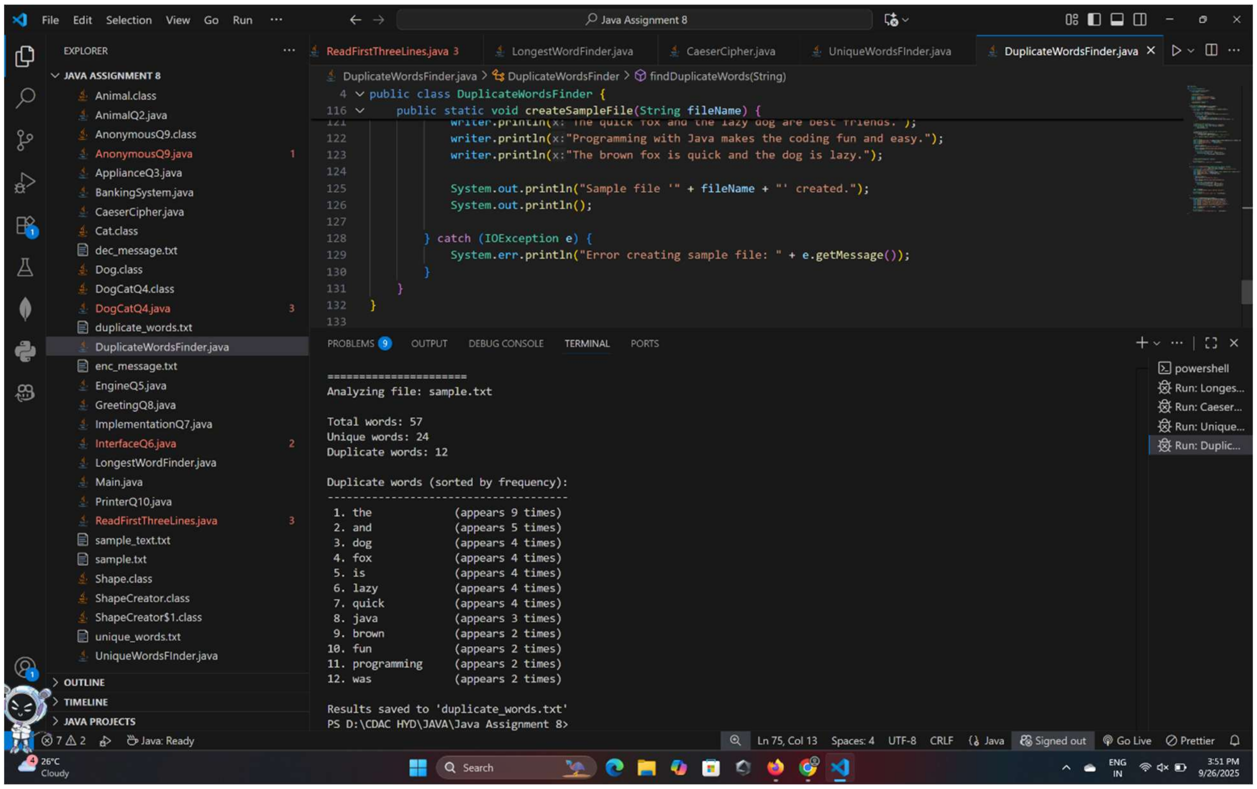
6. Write a program to find unique words in file



```java
public class UniqueWordsFinder {
    public static void createSampleFile(String fileName) {
            writer.println(x:"Programming with Java makes coding easier.");
            writer.println(x:"The quick fox and the lazy dog are friends.");

            System.out.println("Sample file '" + fileName + "' created.");
            System.out.println();

        } catch (IOException e) {
            System.err.println("Error creating sample file: " + e.getMessage());
        }
    }
}
```

```
Sample file 'sample.txt' created.

Unique Words Finder
=================
Analyzing file: sample.txt

Total words found: 41
Unique words count: 22

Unique words:
-------------
 1. and
 2. are
 3. brown
 4. but
 5. coding
 6. dog
 7. easier
 8. fox
 9. friends
10. fun
11. is
12. java
13. jumps
```

7. Write a program to find duplicate words in a file



```java
public class DuplicateWordsFinder {
    public static void createSampleFile(String fileName) {
            writer.println(x:"Programming with Java makes the coding fun and easy.");
            writer.println(x:"The brown fox is quick and the dog is lazy.");

            System.out.println("Sample file '" + fileName + "' created.");
            System.out.println();

        } catch (IOException e) {
            System.err.println("Error creating sample file: " + e.getMessage());
        }
    }
}
```

```
=====================
Analyzing file: sample.txt

Total words: 57
Unique words: 24
Duplicate words: 12

Duplicate words (sorted by frequency):
--------------------------------------
 1. the             (appears 9 times)
 2. and             (appears 5 times)
 3. dog             (appears 4 times)
 4. fox             (appears 4 times)
 5. is              (appears 4 times)
 6. lazy            (appears 4 times)
 7. quick           (appears 4 times)
 8. java            (appears 3 times)
 9. brown           (appears 2 times)
10. fun             (appears 2 times)
11. programming     (appears 2 times)
12. was             (appears 2 times)

Results saved to 'duplicate_words.txt'
PS D:\CDAC HYD\JAVA\Java Assignment 8>
```