

Assignment -11 (24-09-2025)

Roll No - 107

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

The screenshot shows a Java IDE interface with the following details:

- EXPLORER:** Shows files under "JAVA ASSIGNMENT 8" including Animal.class, AnimalQ2.java, AnonymousQ9.class, AnonymousQ9.java, ApplianceQ3.java, BankingSystem.java, Cat.class, Dog.class, DogCatQ4.class, DogCatQ4.java, EngineQ5.java, GreetingQ8.java, ImplementationQ7.java, InterfaceQ6.java, Main.java, PrinterQ10.java, Shape.class, ShapeCreator.class, and ShapeCreator\$1.class.
- EDITOR:** Displays the code for `BankingSystem.java`. The code includes a main method that creates a `Customer` object named `customer4` with specific details and prints it to the console. It also creates an `Account` object named `account5` and sets its account type to "Current".

```
182 public class BankingSystem {
183     public static void main(String[] args) {
184         ...
185         Customer customer4 = new Customer(name:"Bob Johnson", address:"789 Pine Rd, Uptown", customerId:"CUST004", phone:"+1-555-0123");
186         System.out.println(x:"Test Case 4: Customer without Account");
187         customer4.disp();
188         System.out.println();
189
190         System.out.println(x:"Test Case 5: Testing Getters and Setters");
191         Account account5 = new Account();
192         account5.setAccountType(accountType:"Current");
193     }
194 }
```
- PROBLEMS:** Shows 6 errors.
- OUTPUT:** Displays the terminal output of the application's execution. It shows two test cases: one for a normal customer with a positive balance and one for a customer with a negative balance. The output includes customer details and account details.

```
Java Assignment 8_978aled6\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: $1000.0
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
```
- STATUS:** Shows the environment as "Java: Ready".
- NOTIFICATIONS:** Shows a notification for "Rainy days ahead" at 26°C.
- SYSTEM:** Shows system status including battery level, network connection, and date/time (3:25 PM, 9/25/2025).

```

182 public class BankingSystem {
183     public static void main(String[] args) {
184         Customer customer4 = new Customer(name:"Bob Johnson", address:"789 Pine Rd, Uptown", customerId:"CUST004", phone:"123-4567");
185         System.out.println("Test Case 4: Customer without Account");
186         customer4.disp();
187         System.out.println();
188
189         System.out.println("Test Case 5: Testing Getters and Setters");
190         Account account5 = new Account();
191         account5.setAccountType(accountType:"Current");
192
193         Account Details:
194         Account Type: Checking
195         Account Number: CHX002
196         Current Balance: $ 500.0
197         Minimum Balance: $0.0
198         Interest Calculation Error: Cannot calculate interest on negative balance: -500.0
199
200         =====
201
202         Test Case 3: Direct Exception Handling
203         Exception caught: Cannot calculate interest on negative balance: -1000.0
204
205         Test Case 4: Customer without Account
206         === Customer Information ===
207         Customer Name: Bob Johnson
208         Customer ID: CUST004
209         Address: 789 Pine Rd, Uptown
210         Phone Number: +1-555-0789
211
212         No account associated with this customer.
213
214
215         Test Case 5: Testing Getters and Setters
216         === Customer Information ===
217         Customer Name: Alice Wilson
218         Customer ID: CUST005
219         Address: 321 Elm St, Suburb
220         Phone Number: +1-555-0321
221
222         =====
223
224
225         Test Case 4: Customer without Account
226         === Customer Information ===
227         Customer Name: Bob Johnson
228         Customer ID: CUST004
229         Address: 789 Pine Rd, Uptown
230         Phone Number: +1-555-0789
231
232         No account associated with this customer.
233
234
235         Test Case 5: Testing Getters and Setters
236         === Customer Information ===
237         Customer Name: Alice Wilson
238         Customer ID: CUST005
239         Address: 321 Elm St, Suburb
240         Phone Number: +1-555-0321
241
242         Account Details:
243         Account Type: Current
244         Account Number: CUR005
245         Current Balance: $2500.0
246         Minimum Balance: $500.0
247         Annual Interest: $125.00
248
249         =====
250
251         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 to a text file (result.txt)

The screenshot shows the Eclipse IDE interface with the following details:

- File Explorer:** Shows various Java files including Calculator.java, MainToJava.java, CircleJava.java, ShapeToJava.java, MathUtil.java, Problem6.java, and FileAnalyzer.java.
- Code Editor:** Displays the `FileAnalyzer.java` source code. The code reads a file, counts lines, words, and characters, and writes the results to a `result.txt` file.
- Outline View:** Shows the class structure of `FileAnalyzer`.
- Console Output:** Shows the execution of the program. It prints "File analyzed: FileAnalyzer.java", "Lines: 161", "Words: 575", "Characters: 6108", and provides average statistics per line and word. It also indicates to check the `result.txt` file for detailed results.
- System Tray:** Shows a weather icon indicating "Heavier rain In 41 minutes".
- Taskbar:** Shows the date and time as "9/25/2025 3:34:38 PM".

```
File: FileAnalyzer.java
1 package java.assignment;
2
3 import java.io.*;
4 import java.util.*;
5
6 public class FileAnalyzer {
7     int lineCount = 0;
8     int wordCount = 0;
9     int characterCount = 0;
10    String sourceFileName;
11    String resultFileName;
12
13    void main(String[] args) {
14        if (args.length != 1) {
15            System.out.println("Usage: java FileAnalyzer <filename>");
16            System.exit(1);
17        }
18        sourceFileName = args[0];
19        resultFileName = "result.txt";
20
21        try {
22            readAndAnalyzeFile();
23            writeResultsToFile();
24        } catch (IOException e) {
25            System.out.println("Error reading file: " + e.getMessage());
26        }
27    }
28
29    void readAndAnalyzeFile() throws IOException {
30        BufferedReader reader = new BufferedReader(new FileReader(sourceFileName));
31        String line;
32        while ((line = reader.readLine()) != null) {
33            lineCount++;
34            String[] words = line.split("\\s+");
35            wordCount += words.length;
36            characterCount += line.length();
37        }
38        reader.close();
39    }
40
41    void writeResultsToFile() throws IOException {
42        BufferedWriter writer = new BufferedWriter(new FileWriter(resultFileName));
43        writer.printf("File analyzed: %s\n", sourceFileName);
44        writer.printf("Lines: %d\n", lineCount);
45        writer.printf("Words: %d\n", wordCount);
46        writer.printf("Characters: %d\n", characterCount);
47
48        double avgWordsPerLine = (double) wordCount / lineCount;
49        double avgCharactersPerLine = (double) characterCount / lineCount;
50
51        writer.printf("Average words per line: %.2f\n", avgWordsPerLine);
52        writer.printf("Average characters per line: %.2f\n", avgCharactersPerLine);
53
54        writer.close();
55    }
56
57    void displayResults() {
58        System.out.println();
59        System.out.println("ANALYSIS RESULTS:");
60        System.out.println("-----");
61        System.out.println("File analyzed: " + sourceFileName);
62        System.out.println("Lines: " + lineCount);
63        System.out.println("Words: " + wordCount);
64        System.out.println("Characters: " + characterCount);
65
66        System.out.printf("Average words per line: %.2f\n", (double) wordCount / lineCount);
67        System.out.printf("Average characters per line: %.2f\n", (double) characterCount / lineCount);
68
69        System.out.println();
70        System.out.println("Check '" + resultFileName + "' for detailed results.");
71    }
72
73    public static void main(String[] args) {
74        new FileAnalyzer().main(args);
75    }
76}
```

```
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.

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** JAVA - java assignment 9/LineNumberAdder.java - Eclipse IDE
- Left Sidebar (Package Explorer):** Shows the project structure with packages like com.company.accounts, com.company.hr, etc., and files like LineNumberAdder.java.
- Central Area (Code Editor):** Displays the Java code for LineNumberAdder.java. The code reads a source file, adds line numbers to each line, and writes the result to an output file. It handles exceptions for file not found and I/O errors.
- Right Sidebar (Outline):** Shows the class hierarchy for LineNumberAdder.
- Bottom Status Bar:** Shows the terminal output, which includes progress messages like "Processed 9 lines..." and "Total lines processed: 169", followed by "SUCCESS! Line numbers added successfully!".
- System Tray:** Shows weather information: 26°C Light rain.

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

The screenshot shows the VS Code interface with the following details:

- Title Bar:** Java Assignment 8
- Left Sidebar (Explorer):** Shows the project structure with Java files like GreetingQ8.java, AnonymousQ9.java, ReadFirstThreeLines.java, etc.
- Central Area (Code Editor):** Displays the Java code for ReadFirstThreeLines.java. It uses a Scanner to read the first three lines of a file named fileName. It prints the lines to the console and handles cases where the file has fewer than three lines.
- Bottom Status Bar:** Shows the terminal output, which includes the first three lines of a sample.txt file: "Line 1: This is the first line of the sample file.", "Line 2: Here comes the second line with some text.", and "Line 3: The third line contains more sample content."
- System Tray:** Shows weather information: Heavy rain Today.

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

Java Assignment 8

File Edit Selection View Go Run ...

EXPLORER

JAVA ASSIGNMENT 8

- Animal.class
- AnimalQ2.java
- AnonymousQ9.class
- AnonymousQ9.java
- ApplianceQ3.java
- BankingSystem.java
- Cat.class
- Dog.class
- DogCatQ4.class
- DogCatQ4.java
- EngineQ5.java
- GreetingQ8.java
- ImplementationQ7.java
- InterfaceQ6.java
- LongestWordFinder.java
- Main.java
- PrinterQ10.java
- ReadFirstThreeLines.java
- sample_text.txt
- sample.txt
- Shape.class
- ShapeCreator.class
- ShapeCreator\$1.class

LongestWordFinder.java > LongestWordFinder

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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

powershell

Run: LongestWordFinder

Ln 288, Col 5 Spaces: 4 UTF-8 CRLF Java Signed out Go Live Prettier

7 2 8 Java: Ready

Top Stories Ba***ds of Boly...

Search

3:00 PM 9/26/2025

Java Assignment 8

File Edit Selection View Go Run ...

EXPLORER

JAVA ASSIGNMENT 8

- Animal.class
- AnimalQ2.java
- AnonymousQ9.class
- AnonymousQ9.java
- ApplianceQ3.java
- BankingSystem.java
- Cat.class
- Dog.class
- DogCatQ4.class
- DogCatQ4.java
- EngineQ5.java
- GreetingQ8.java
- ImplementationQ7.java
- InterfaceQ6.java
- LongestWordFinder.java
- Main.java
- PrinterQ10.java
- ReadFirstThreeLines.java
- sample_text.txt
- sample.txt
- Shape.class
- ShapeCreator.class
- ShapeCreator\$1.class

LongestWordFinder.java > LongestWordFinder

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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 ==

Word Length	Count	Percentage
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%)

powershell

Run: LongestWordFinder

Ln 288, Col 5 Spaces: 4 UTF-8 CRLF Java Signed out Go Live Prettier

7 2 8 Java: Ready

26°C Cloudy

Search

3:00 PM 9/26/2025

```
LongestWordFinder.java > LongestWordFinder
6  public class LongestWordFinder {
289   private static void createSampleFileIfNotExists() {
290     try (PrintWriter writer = new PrintWriter(new FileWriter(DEFAULT_FILE))) {
291       writer.println("The quick brown fox jumps over the lazy dog.");
292       writer.println("Java programming is extraordinarily powerful and versatile.");
293       writer.println("Supercalifragilisticexpialidocious is a very long word!");
294       writer.println("Short words: a, an, the, is, in, on, at, to, of, for.");
295       writer.println("Programming languages include: Java, Python, JavaScript, C++.");
296       writer.println("This file contains various words of different lengths.");
297       writer.println("Some technical terms: algorithm, implementation, optimization.");
298       writer.println("Antidisestablishmentarianism is another extremely long word.");
299     }
300   }
301   System.out.println("Sample file " + DEFAULT_FILE + " created for demonstration.");
302   System.out.println();
303 }
304
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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 implement Caeser cipher using files.

```
CaeserCipher.java > CaeserCipher
3  public class CaeserCipher{
4    private static final int DISPLACEMENT = 3;
5
6    Run | Debug
7    public static void main(String[] args) {
8      System.out.println("Caesar Cipher Program (Displacement = 3)");
9      System.out.println("=====");
10
11      String originalMessage = "Hello World!\n" +
12          "This is a secret message.\n" +
13          "Caesar cipher with displacement 3.\n" +
14          "Java programming is fun!";
15
16      System.out.println("Original Message:");
17
18      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\dee2c8069480bee3442a5fed4fd3d9\redhat.java\jdt_ws\Java Assignment 8_978a1edc\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!
Wkiv lv d vhfuw phvvdjh.
Fdhdvu flskhu zlwk glvsodfhpqz.
Mydjd surjudplajj 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

The screenshot shows a Java assignment titled "Java Assignment 8" in a code editor. The left sidebar lists various Java files under the "JAVA ASSIGNMENT 8" category. The main editor area displays the content of "UniqueWordsFinder.java". The code creates a sample file named "sample.txt" containing the text "Programming is fun and Java is powerful.", prints a message to the console, and handles an IOException. The terminal below shows the execution of the program, which finds 41 unique words in the sample file.

```
public class UniqueWordsFinder {
    public static void createSampleFile(String fileName) {
        Writer writer = null;
        try {
            writer = new PrintWriter(fileName);
            writer.println("Programming is fun and Java is powerful.");
            writer.println("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());
        }
    }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
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

The screenshot shows the Java Assignment 8 project in the Code Editor. The Explorer sidebar lists various Java files and resources. The DuplicateWordsFinder.java file is open in the editor, showing code to create a sample file with sample text. The Problems panel shows one warning about an unused import. The terminal output shows the analysis of a sample.txt file, displaying word counts and a frequency sorted list of duplicate words.

```
public class DuplicateWordsFinder {
    public static void createSampleFile(String fileName) {
        writer.println("The quick fox and the lazy dog are best friends.");
        writer.println("Programming with Java makes the coding fun and easy.");
        writer.println("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());
    }
}
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
=====
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'