**CSC262 Java Programming – Module 4 Assignment**

|  |  |
| --- | --- |
| **First Name** | Zachary |
| **Last Name** | Banks |
| **ID#** | 900648740 |
| **Assignment Week#** | 4 |
| **Email Address** | z.banks8740@student.nu.edu |

# How to submit your Assignment

After filling all the parts in this file, please follow the following steps.

1. Add your name and ID to the first page.
2. Save the file in the original format (Docx or Doc)

(Please **do not** convert to other file formats e.g. PDF, ZIP, RAR, …).

1. Rename the file as

*YOUR* ***First*** *Name - YOUR* ***Last*** *Name- YOUR student ID-* CSC262*.docx*

**Example:**

John – Smith - 234566435 – CSC262.docx

1. Upload the file and submit it (only using Blackboard)

Please do not hesitate to contact your instructor for CSC262, should you have any questions.

**Instructions:**

This signature assignment is a group project. Teamwork and collaboration will be essential in completing this assignment.  All team members must remain engaged and actively participate in the group meetings and communications while contributing to the design and implementation of the solution. In this Signature Assignment, you will demonstrate your mastery of Java concepts covered in the course. (80 points for the programming)

Please note that each member will be peer evaluated by the other team members according to the following team building criteria . The average peer evaluation scores from the other team members will be used as your teamwork grade. (20 points for peer evaluation).

# Problem 1: Expenses Organizer

A user records their expenses in a text file, where each line contains the following fields separated by semicolons: the category of the expense (e.g., Groceries, Utilities, Entertainment, Medical etc.), a brief description, the amount spent, and the date of the expense.

Write a Java program that reads this file and calculates the total amount spent for each expense category. Write a separate file for each expense category. Each expense category file should contain the entries for that category. Name the output files Groceries.txt, Utilities.txt, and so on. The program should also handle errors gracefully by displaying an appropriate message if the file does not exist or if the format of any line is incorrect.

Sample input.txt

A close-up of a computer screen

AI-generated content may be incorrect.

Output:

1. Upload the screenshot of the input.txt and each expense category files.
2. Upload output screenshot for the Exceptions handler.

|  |
| --- |
| **Your code for this problem** |
| **package** csc262.assignment4.io;    2. **import** java.io.\*; 3. **import** java.util.\*; 5. /\*\* 6. \* Problem 1: Expenses Organizer 7. \* CSC262 - Assignment 4 8. \* Group Members: Zach Banks, Jess Scott 9. \* 10. \* This program reads an expenses file with entries in the format: 11. \* Category;Description;Amount;Date 12. \* It categorizes each entry and writes a separate file for each expense category. 13. \* It also handles missing file exceptions and malformed input lines. 14. \*/ 15. **public** **class** Problem1\_ExpensesOrganizer { 17. **public** **static** **void** main(String[] args) { 18. Map<string, list<string="">> categorizedExpenses = **new** HashMap<>(); 19. File inputFile = **new** File("input.txt"); 21. **try** (Scanner scanner = **new** Scanner(inputFile)) { 22. **while** (scanner.hasNextLine()) { 23. String line = scanner.nextLine(); 24. String[] parts = line.split(";"); 25. **if** (parts.length != 4) { 26. System.out.println("Invalid line format: " + line); 27. **continue**; 28. } 30. String category = parts[0].trim(); 31. String entry = String.join(";", Arrays.copyOfRange(parts, 1, 4)); 33. categorizedExpenses 34. .computeIfAbsent(category, k -> **new** ArrayList<>()) 35. .add(entry); 36. } 38. // Write categorized entries to separate files 39. **for** (Map.Entry<string, list<string="">> entry : categorizedExpenses.entrySet()) { 40. String category = entry.getKey(); 41. List<string> entries = entry.getValue(); 43. **try** (PrintWriter writer = **new** PrintWriter(**new** FileWriter(category + ".txt"))) { 44. **for** (String expense : entries) { 45. writer.println(expense); 46. } 47. } 48. } 50. System.out.println("Expense files created successfully."); 52. } **catch** (FileNotFoundException e) { 53. System.out.println("Error: input.txt not found."); 54. } **catch** (IOException e) { 55. System.out.println("Error writing category file: " + e.getMessage()); 56. } 57. } 58. }</string></string,></string,> |

Run the code and insert the result in the following box.

|  |
| --- |
| **Sample Run Result** |
|  |

# Problem 2: Exception Handler

In chapter 7, we studied the demonstration code DataAnalyzer.java (on page 361). Modify the DataAnalyzer.java program so that you do not call hasNextInt or hasNextDouble. Simply have nextInt and nextDouble throw a NoSuchElementException and catch it in the main method.

Requirement: Please create three input files with the following contents and use these files to test your exception handler:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

|  |
| --- |
| **Your code for this problem** |
| **package** csc262.assignment4.io;    2. **import** java.io.\*; 3. **import** java.util.\*; 5. /\*\* 6. \* Problem 2: DataAnalyzer with Exception Handling 7. \* CSC262 - Assignment 4 8. \* Group Members: Zach Banks, Jess Scott 9. \* 10. \* This program reads a file containing a count and several doubles. 11. \* It uses nextInt and nextDouble without calling hasNext, and catches 12. \* NoSuchElementException in main as required. 13. \*/ 14. **public** **class** Problem2\_DataAnalyzer { 16. **public** **static** **void** main(String[] args) { 17. Scanner console = **new** Scanner(System.in); 18. **boolean** done = **false**; 20. **while** (!done) { 21. **try** { 22. System.out.print("Enter filename: "); 23. String filename = console.nextLine(); 24. **double**[] data = readFile(filename); 26. System.out.println("Data loaded:"); 27. **for** (**double** d : data) { 28. System.out.println(d); 29. } 31. done = **true**; 32. } **catch** (FileNotFoundException e) { 33. System.out.println("File not found."); 34. } **catch** (NoSuchElementException e) { 35. System.out.println("File contents invalid."); 36. } **catch** (IOException e) { 37. e.printStackTrace(); 38. } 39. } 40. } 42. **public** **static** **double**[] readFile(String filename) **throws** IOException { 43. File inFile = **new** File(filename); 44. **try** (Scanner in = **new** Scanner(inFile)) { 45. **return** readData(in); 46. } 47. } 49. **public** **static** **double**[] readData(Scanner in) **throws** IOException { 50. **int** numberOfValues = in.nextInt(); // May throw NoSuchElementException 51. **double**[] data = **new** **double**[numberOfValues]; 53. **for** (**int** i = 0; i < numberOfValues; i++) { 54. data[i] = in.nextDouble(); // May throw NoSuchElementException 55. } 57. **if** (in.hasNext()) { 58. **throw** **new** IOException("End of file expected"); 59. } 61. **return** data; 62. } 63. } |

Run the code and insert the result in the following box.

|  |
| --- |
| **Sample Run Result** |
|  |

**The end**