File IO

Problem 1: Copy File Content

Requirements:

1. Read the content from `source.txt`.
2. Write the content to `destination.txt`.

Approach:

1. Use `FileReader` to read from `source.txt`.
2. Use `FileWriter` to write to `destination.txt`.
3. Handle exceptions and ensure resources are properly closed using try-with-resources.

Problem 2: Read and Display File Content

Requirements:

1. Read the content from `input.txt`.
2. Print each line of the file to the console.

Approach:

1. Use `BufferedReader` to read from `input.txt`.
2. Read lines using a loop and print each line to the console.
3. Handle exceptions and ensure the reader is properly closed.

Problem 3: Write User Input to a File

Requirements:

1. Read user input from the console.
2. Write the input to `user_input.txt`.

Approach:

1. Use `Scanner` to read user input.
2. Use `FileWriter` to write the input to `user_input.txt`.
3. Handle exceptions and ensure resources are properly closed.

Problem 4: Count Words in a File

Requirements:

1. Read the content from `text_file.txt`.
2. Count the number of words in the file.
3. Print the word count to the console.

Approach:

1. Use `BufferedReader` to read from `text_file.txt`.
2. Split each line into words and count the total number of words.
3. Print the word count and handle exceptions.

Problem 5: Reverse File Content

Requirements:

1. Read the content from `input.txt`.
2. Reverse the content.
3. Write the reversed content to `reversed.txt`.

Approach:

1. Use `BufferedReader` to read from `input.txt`.
2. Use `StringBuilder` to reverse the content.
3. Use `BufferedWriter` to write the reversed content to `reversed.txt`.
4. Handle exceptions and ensure all resources are properly closed.

Buffer Reader and Buffer Writer (Any two)

Problem 1: Large Data File Processing

Scenario:

You are tasked with processing a large data file containing records of customer transactions. Each line in the file contains a transaction record with fields such as Transaction ID, Customer ID, Transaction Amount, and Date. The goal is to filter out transactions above a certain amount and write these filtered records to a new file.

Requirements:

1. **Read Transactions:**
   - Read records from `transactions.txt`.
   - Each record is in the format: `TransactionID,CustomerID,TransactionAmount,Date`.
2. **Filter Transactions:**
   - Filter transactions where `TransactionAmount` is greater than a specified threshold.
3. **Write Filtered Data:**
   - Write the filtered transactions to `filtered_transactions.txt`.

Implementation Approach:

1. **Read and Process with `BufferedReader`:**
   - Use `BufferedReader` to efficiently read lines from `transactions.txt`.
   - Split each line to extract transaction details and filter based on the amount.
2. **Write with `BufferedWriter`:**
   - Use `BufferedWriter` to write the filtered records to `filtered_transactions.txt`.

Problem 2: Aggregating Report Data

Scenario:

You are tasked with creating a summary report from a file containing log entries. Each line in the log file represents an entry with fields such as Date, Log Level, and Message. The goal is to count the occurrences of each log level and write the summary report to a new file.

Requirements:

1. **Read Log Entries:**
   - Read log entries from `logs.txt`.
   - Each entry is in the format: `Date,LogLevel,Message`.
2. **Count Log Levels:**
   - Count occurrences of each `LogLevel`.
3. **Write Summary Report:**
   - Write the log level counts to `log_summary.txt`.

Implementation Approach:

1. **Read and Process with `BufferedReader`:**
   - Use `BufferedReader` to read lines from `logs.txt`.
   - Use a `HashMap` to count occurrences of each log level.
2. **Write with `BufferedWriter`:**
   - Use `BufferedWriter` to write the counts to `log_summary.txt`.

Problem 3: Copying File Contents

Scenario:

You need to create a utility that copies the contents of one text file to another. This utility should be able to handle large files efficiently.

Requirements:

1. **Read Source File:**
   - Read data from `source.txt`.
2. **Write to Destination File:**
   - Write the copied data to `destination.txt`.

Implementation Approach:

1. **Read and Write with `BufferedReader` and `BufferedWriter`:**
   - Use `BufferedReader` to read from `source.txt`.
   - Use `BufferedWriter` to write to `destination.txt`.

Problem 4: Merging Multiple Files

Scenario:

You need to merge the contents of multiple text files into a single file. The files contain customer feedback and need to be combined for further analysis.

Requirements:

1. **Read from Multiple Files:**
   - Read data from `file1.txt`, `file2.txt`, and `file3.txt`.
2. **Write to Merged File:**
   - Write the combined data to `merged_feedback.txt`.

Implementation Approach:

1. **Read from Multiple Files with `BufferedReader`:**
   - Use `BufferedReader` to read each file.
2. **Write Combined Data with `BufferedWriter`:**
   - Use `BufferedWriter` to write to `merged_feedback.txt`.