WordCount Client-Server Solution Documentation

Overview

This Java 8 solution implements a parallel client-server architecture using sockets to stream the contents of two large text files (Dracula and Frankenstein).

Each server serves one file and terminates after a single client request. The client connects to both servers concurrently, processes the incoming lines,

and computes the top 5 most common words.

Key objectives met:

- Thread safety and scalability using concurrent structures.
- Clean code separation by responsibility.
- Automatic resource cleanup (no socket leaks).
- Enum-based configuration to avoid magic numbers.

Why This Approach

- Simplicity: Uses Java core features only (no frameworks).
- Portability: Uses classpath resources for file loading.
- Thread safety: ConcurrentHashMap and ExecutorService ensure safe parallel word counting.
- Maintainability: Clear package structure, meaningful method names, enums for configuration.
- Flexibility: Servers shut down cleanly after serving one client, client processes both servers concurrently.

Thread Safety

Thread safety is achieved in the WordCounter class using ConcurrentHashMap.

We use the merge() method which is atomic and handles concurrent updates safely.

Example:

wordCount.merge(word, 1, Integer::sum);

This ensures that even if multiple threads call countWords at the same time, updates won't conflict.

Clean Code Principles Applied

- SRP (Single Responsibility Principle): Each class does exactly one job.

- DRY (Don't Repeat Yourself): Reused logic for reading, counting, and connection handling.

- Enum instead of magic numbers: Port enum for clarity and safety.

- try-with-resources: Ensures sockets and streams are closed automatically.

- Readable method and variable names.

Component Documentation

1. Exec.java

- Starts two FileServer threads and the client.

- Waits, coordinates, and shuts down servers after processing.

2. FileServer.java

- Accepts one connection and hands it to ClientHandler.

- Shuts down gracefully.

3. ClientHandler.java

- Reads a file line-by-line from classpath and sends it to the client.

- Closes the socket after sending.

4. WordCountClient.java

- Initializes server connections using ExecutorService.

- Aggregates word counts and prints top 5.

5. ServerReader.java

- Opens socket to server, reads lines, passes to WordCounter.

6. WordCounter.java

- Thread-safe word count logic using ConcurrentHashMap.

- Outputs the most common N words.

7. ServerInfo.java

- Data structure for host and Port enum.

8. Port.java

- Enum to define port numbers instead of hardcoding them.