Software Design Document (SDD) - Book Word Counter

1. Introduction

The Book Word Counter is a Java application that reads a `.txt` file, counts the occurrences of each word, and outputs the results in alphabetical order. The program removes punctuation, normalizes words to lowercase, and allows users to query word counts interactively via the command line. It saves the final count in a generated output file named WordCountResults-BookName.txt. It is designed to handle large text files efficiently using Java I/O and HashMap for word storage.

2. System Architecture

The system follows a modular design:

- `Main.java`: Serves as the entry point, initializes `BookWordCounter`, and calls its methods.
- `BookWordCounter.java`: Contains the core logic for file reading, processing, and saving results.
- `HashMap<String, Integer>` is used to store words and their occurrences.
- `BufferedReader` is used for efficient reading of large files.

Data Flow:

1. `Main.java` calls `processBook(filePath)`.
2. `processBook()` reads the file line by line.
3. Each line is passed to `processLine(line)`, which:
   - Removes punctuation.
   - Converts text to lowercase.
   - Splits words and updates their count in `HashMap`.
4. `saveResults(outputFilePath)` writes sorted results to a named output file (WordCountResults-BookName.txt).
5. Users can query word counts interactively through the CLI after processing.

3. Algorithms & Data Structures

- File Reading: Uses `BufferedReader` for efficient line-by-line reading (`O(n)` complexity).
- Word Storage: Uses `HashMap<String, Integer>` for word counting (`O(1)` average insertion time).
- Sorting: Uses `Collections.sort()` to sort words alphabetically (`O(n log n)` complexity).

Performance:

Time Complexity: `O(n log n)`, where `n` is the number of unique words (due to sorting).

Space Complexity: `O(U)`, where `U` is the number of unique words in the text.

4. UML Diagram