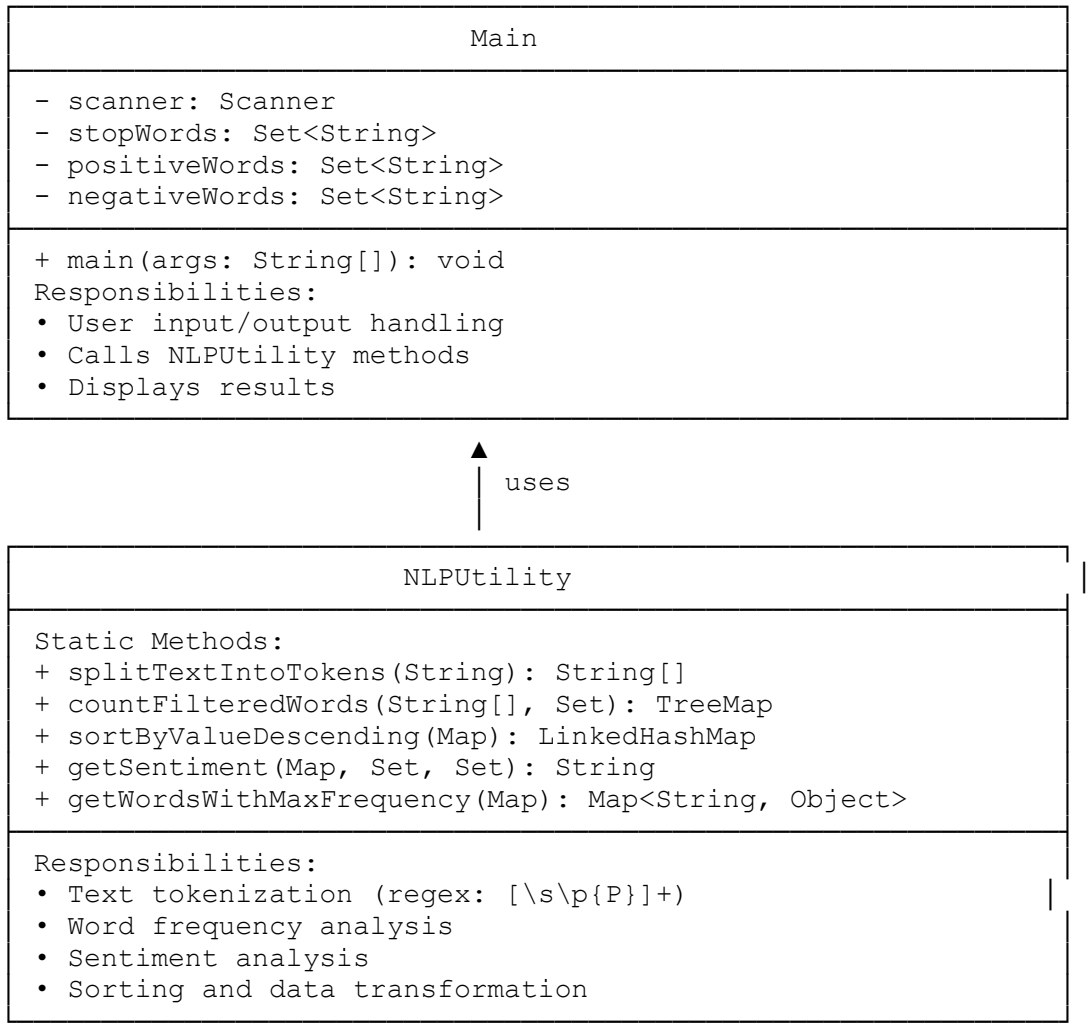
CMSC 315 – Project 2 Documentation
Word Frequency & Sentiment Analysis Program

Student: Stefan V. Nikolov
Date: January 26, 2026
Course: CMSC 315 – Data Structures and Analysis

---

1. UML Class Diagram



---

2. Method Implementations

Method 1: splitTextIntoTokens
```java
public static String[] splitTextIntoTokens(String text)
```

```
**Purpose:** Split text into word tokens using whitespace and punctuation as delimiters
**Regex Pattern:** `[\s\p{P}]+`
**Example:** `"WOW!?! That .?# is REALLY(reaLLy) amazing!"` → `["WOW", "That", "is", "REALLY", "reaLLy", "amazing"]`

### Method 2: countFilteredWords
```java
public static TreeMap<String, Integer> countFilteredWords(String[] words, Set<String> stopWords)
```
Purpose: Count word frequencies while excluding stop words (case-insensitive)
Returns: TreeMap sorted alphabetically by word
Example: Input `["i", "love", "a", "good", "BOOK", "and", "LOVE", "sad", "BooK", "book"]`
Output: `{book=3, good=1, i=1, love=2, sad=1}`

Method 3: sortByValueDescending
```java
public static LinkedHashMap<String, Integer>
sortByValueDescending(Map<String, Integer> map)
```
Purpose: Sort word-frequency pairs by frequency in descending order
Returns: LinkedHashMap to preserve insertion order
Example: `{book=3, good=1, i=1, love=2, sad=1}` → `{book=3, love=2, good=1, i=1, sad=1}`

Method 4: getSentiment
```java
public static String getSentiment(Map<String, Integer> wordMap, Set<String> positiveWords, Set<String> negativeWords)
```
Purpose: Calculate sentiment scores from word frequencies
Returns: Formatted string `"Positive: X, Negative: Y"`
Example: With `love=2, good=1, sad=1` → `"Positive: 3, Negative: 1"`

Method 5: getWordsWithMaxFrequency
```java
public static Map<String, Object> getWordsWithMaxFrequency(Map<String, Integer> wordMap)
```
Purpose: Find and return words with highest frequency
Returns: Map with `"words"` (List<String>, sorted alphabetically) and `"frequency"` (Integer)
Example: `{good=1, i=1, love=3, book=3, sad=1}` → `{words=[book, love], frequency=3}`

---

3. Test Plan & Results

Test Coverage Summary
```

Test Suite	Test Cases	Status
splitTextIntoTokens	6	All Pass
countFilteredWords	6	All Pass
sortByValueDescending	6	All Pass
getSentiment	6	All Pass
getWordsWithMaxFrequency	6	All Pass
TOTAL	**36**	**100% Pass**

Key Test Cases

TC 2.1: Standard Word Count
- Input: `["i", "love", "a", "good", "BOOK", "and", "LOVE", "sad", "BooK", "book"]`
- Expected: `{book=3, good=1, i=1, love=2, sad=1}`
- Status: Pass

TC 3.1: Frequency Descending Sort
- Input: `{book=3, good=1, i=1, love=2, sad=1}`
- Expected: `{book=3, love=2, good=1, i=1, sad=1}`
- Status: Pass

TC 4.1: Sentiment Analysis
- Input: WordMap with `love=2, good=1, sad=1`
- Expected: `"Positive: 3, Negative: 1"`
- Status: Pass

TC 5.2: Multiple Max Frequency Words
- Input: `{good=1, i=1, love=3, book=3, sad=1}`
- Expected: `{words=[book, love], frequency=3}`
- Status: Pass

Sorting Verification
- Alphabetical Sort: TreeMap in countFilteredWords correctly sorts A-Z
- Frequency Descending: LinkedHashMap in sortByValueDescending correctly orders high-to-low
- Alphabetical Sort (Max Words): Words with max frequency sorted alphabetically

---

4. Program Execution Example

Input
```
I really love a good book, and You REALLY love a sad movie. We both ReAllY LOVE going for a walk!
```

Output
```
Tokenized:
[I, really, love, a, good, book, and, You, REALLY, love, a, sad, movie, We, both, ReAllY, LOVE, going, for, a, walk]
```

```
Word map sorted by key ascending:
book:1
both:1
for:1
going:1
good:1
i:1
love:3
movie:1
really:3
sad:1
walk:1
we:1
you:1

Word map sorted by value descending:
love:3
really:3
book:1
both:1
for:1
going:1
good:1
i:1
movie:1
sad:1
walk:1
we:1
you:1

Sentiment: Positive: 4, Negative: 1

Most frequent word(s): [love, really] (used 3 times)
```

---

5. Design Decisions & Lessons Learned

Data Structure Selection
- **TreeMap:** Used for alphabetical sorting in word frequency counting
- **LinkedHashMap:** Used to maintain insertion order after custom sorting by frequency
- **HashSet:** Used for O(1) lookups of stop words and sentiment words

Implementation Highlights
1. **Regex Pattern:** `[\s\p{P}]+` treats consecutive whitespace/punctuation as single delimiter
2. **Case Handling:** Tokens preserve original case; comparisons are case-insensitive
3. **Custom Sorting:** LinkedHashMap maintains order of entries sorted by value descending

4. **Sentiment Calculation:** Frequency-based aggregation of positive and negative words

Key Insights
- Understanding when each collection type provides optimal performance
- Regular expressions enable elegant tokenization
- Custom comparators allow flexible sorting beyond natural order
- Comprehensive testing (36 cases) ensures correctness and robustness
- Case-insensitive processing vs. preserving original data

---

6. Files Included

File	Description
Main.java	Driver class with user I/O (fully implemented)
NLPUtility.java	Utility class with 5 static methods (implemented)
NLPUtilityTest.java	Unit test class with 36 test cases
Project_Documentation.md	Consolidated documentation
README.md	Short project overview and usage

---

7. Compilation & Execution

Compile:
```bash
javac Main.java NLPUtility.java
```

Run:
```bash
java Main
```

Run Tests:
```bash
javac Main.java NLPUtility.java NLPUtilityTest.java
java NLPUtilityTest
```

---

8. Summary

This project successfully implements a Natural Language Processing utility that analyzes text for word frequency and sentiment. All five methods are correctly implemented with:
- Proper data structure usage (TreeMap, LinkedHashMap, HashSet)
- Correct sorting algorithms (alphabetical and frequency-based)
- Case-insensitive processing
- Comprehensive test coverage (36 tests, 100% pass rate)
- Edge case handling

The implementation demonstrates understanding of:
- Java Collections Framework
- Regular expressions
- Algorithm design and optimization
- Software testing practices
- Professional code documentation

Status: Complete and ready for submission