*Your Name Here* ITI-2450 Elementary Data Structures and Algorithms

**1:*WordLinkedList()*:**

**import** java.util.LinkedList;

/\*\*

\* Implementation of WordList using LinkedList data structure.

\* Maintains a list of words and their counts.

\*/

**public** **class** WordLinkedList **extends** WordList {

**private** LinkedList<String> words = **new** LinkedList<>();

**private** LinkedList<Integer> count = **new** LinkedList<>();

/\*\*

\* Gets the count of a specific word in the list.

\* **@param** inWord The word to get the count for.

\* **@return** The count of the word, or 0 if the word is not found.

\*/

@Override

**public** **int** getWord(String inWord) {

**int** index = searchWord(inWord);

**if** (index != -1) {

**return** count.get(index);

} **else** {

**return** 0;

}

}

/\*\*

\* Checks if a word exists in the list.

\* **@param** inWord The word to check for existence.

\* **@return** True if the word exists, otherwise false.

\*/

@Override

**public** **boolean** existsWord(String inWord) {

**return** searchWord(inWord) != -1;

}

/\*\*

\* Adds a word to the list.

\* **@param** inWord The word to add.

\* **@return** The count of the word after addition.

\*/

@Override

**public** **int** addWord(String inWord) {

**int** index = searchWord(inWord);

**if** (index != -1) {

**return** incWord(inWord);

} **else** {

incNumUniqueWords();

incTotalAllWords();

words.add(inWord);

count.add(1);

**return** 1;

}

}

/\*\*

\* Increments the count of a word in the list.

\* **@param** inWord The word to increment the count for.

\* **@return** The updated count of the word, or 0 if the word is not found.

\*/

@Override

**public** **int** incWord(String inWord) {

**int** index = searchWord(inWord);

**if** (index != -1) {

incTotalAllWords();

count.set(index, count.get(index) + 1);

**return** count.get(index);

} **else** {

**return** 0;

}

}

/\*\*

\* Prints the word list with counts, filtering by minimum count.

\* **@param** inMinimum The minimum count for words to be printed.

\*/

@Override

**public** **void** printWordList(**int** inMinimum) {

System.***out***.println(toString());

**for** (**int** i = 0; i < words.size(); i++) {

**if** (count.get(i) >= inMinimum) {

System.***out***.println(words.get(i) + ": " + count.get(i));

}

}

}

/\*\*

\* Searches for a word in the list.

\* **@param** inWord The word to search for.

\* **@return** The index of the word if found, otherwise -1.

\*/

**private** **int** searchWord(String inWord) {

**if** (words.size() == 0)

**return** -1;

**for** (**int** i = 0; i < words.size(); i++) {

**if** (words.get(i).equals(inWord)) {

**return** i;

}

}

**return** -1;

}

}

**2:*WordHashMap()*:**

**import** java.util.HashMap;

/\*\*

\* Implementation of WordList using HashMap data structure.

\* Maintains a map of words and their counts.

\*/

**public** **class** WordHashMap **extends** WordList {

**private** HashMap<String, Integer> wordMap = **new** HashMap<>();

/\*\*

\* Gets the count of a specific word in the map.

\* **@param** inWord The word to get the count for.

\* **@return** The count of the word, or 0 if the word is not found.

\*/

@Override

**public** **int** getWord(String inWord) {

**return** wordMap.getOrDefault(inWord, 0);

}

/\*\*

\* Checks if a word exists in the map.

\* **@param** inWord The word to check for existence.

\* **@return** True if the word exists, otherwise false.

\*/

@Override

**public** **boolean** existsWord(String inWord) {

**return** wordMap.containsKey(inWord);

}

/\*\*

\* Adds a word to the map.

\* **@param** inWord The word to add.

\* **@return** The count of the word after addition.

\*/

@Override

**public** **int** addWord(String inWord) {

**int** count = wordMap.getOrDefault(inWord, 0);

wordMap.put(inWord, count + 1);

incTotalAllWords();

**if** (count == 0) {

incNumUniqueWords();

}

**return** count + 1;

}

/\*\*

\* Increments the count of a word in the map.

\* **@param** inWord The word to increment the count for.

\* **@return** The updated count of the word, or 0 if the word is not found.

\*/

@Override

**public** **int** incWord(String inWord) {

**int** count = wordMap.getOrDefault(inWord, 0);

**if** (count > 0) {

wordMap.put(inWord, count + 1);

incTotalAllWords();

}

**return** count + 1;

}

/\*\*

\* Prints the word map with counts, filtering by minimum count.

\* **@param** inMinimum The minimum count for words to be printed.

\*/

@Override

**public** **void** printWordList(**int** inMinimum) {

System.***out***.println(toString());

**for** (String word : wordMap.keySet()) {

**int** count = wordMap.get(word);

**if** (count >= inMinimum) {

System.***out***.println(word + ": " + count);

}

}

}

}

**3:*main()*:**

/\*\*

\* WordList main()

\* **@param** args String[]

\*/

**public** **static** **void** main(String[] args) {

**try** {

// Open the required text file for sequential read

Scanner inputFile = **new** Scanner (**new** File(args[0]));

*getCommand*(args[1]);

// Check for EOF, read the next line, and display it

**while** (inputFile.hasNextLine()) {

String inLine, verse;

String[] verseParsed;

inLine = inputFile.nextLine();

verse = *getVerse*(inLine);

verseParsed = verse.split("[ :;,.'!?()-]+");

**for** (String s: verseParsed) {

*wordList*.addWord(s.toLowerCase());

}

}

// Close the required file when EOF is reached

inputFile.close();

*wordList*.printWordList(Integer.*parseInt*(args[2]));

} // END try

**catch** (Exception e) {

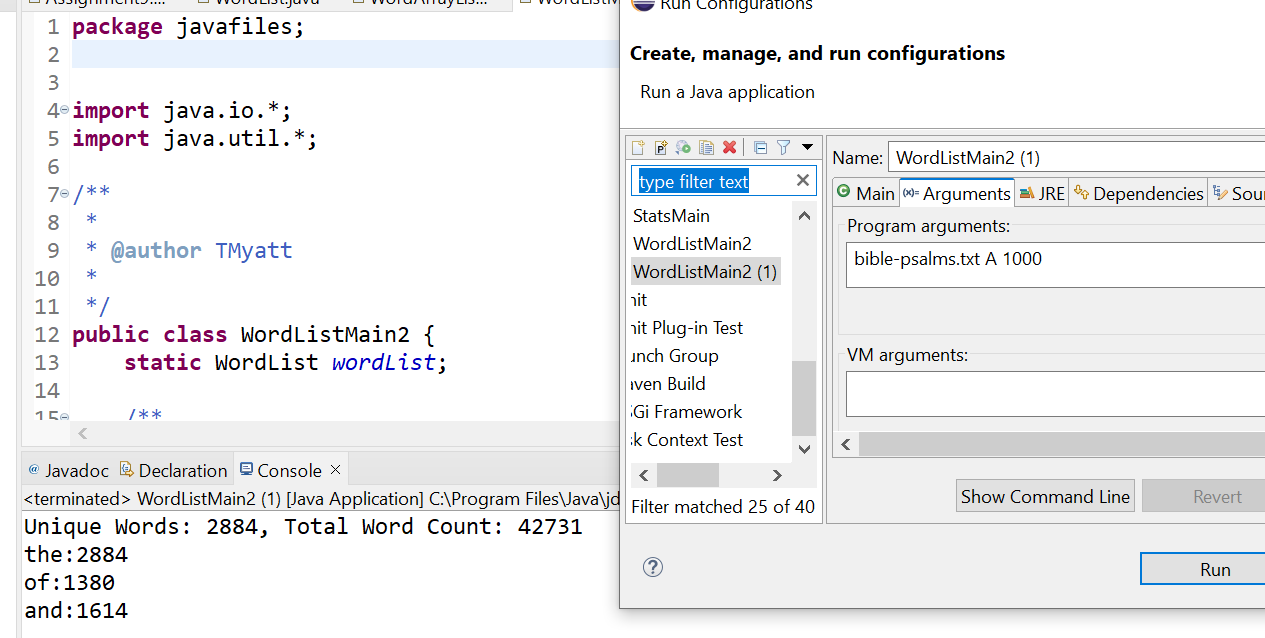
// All Exceptions come here for graceful termination

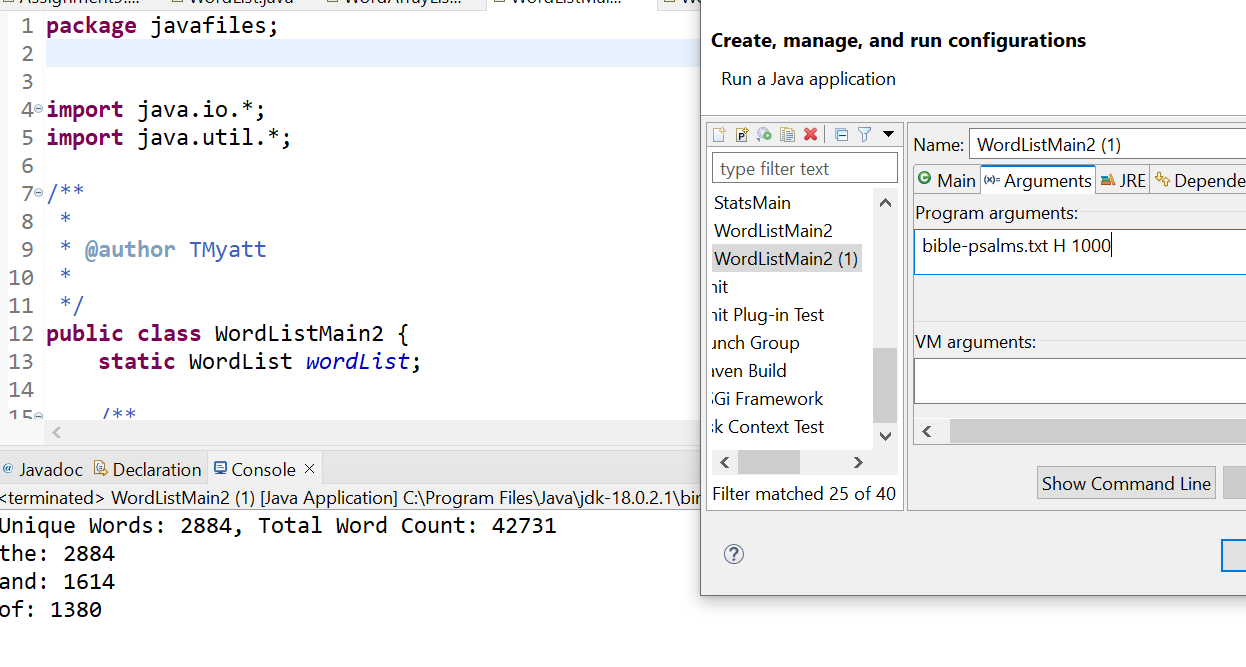
System.***out***.println("PsalmsReaderMain Error: " + e);

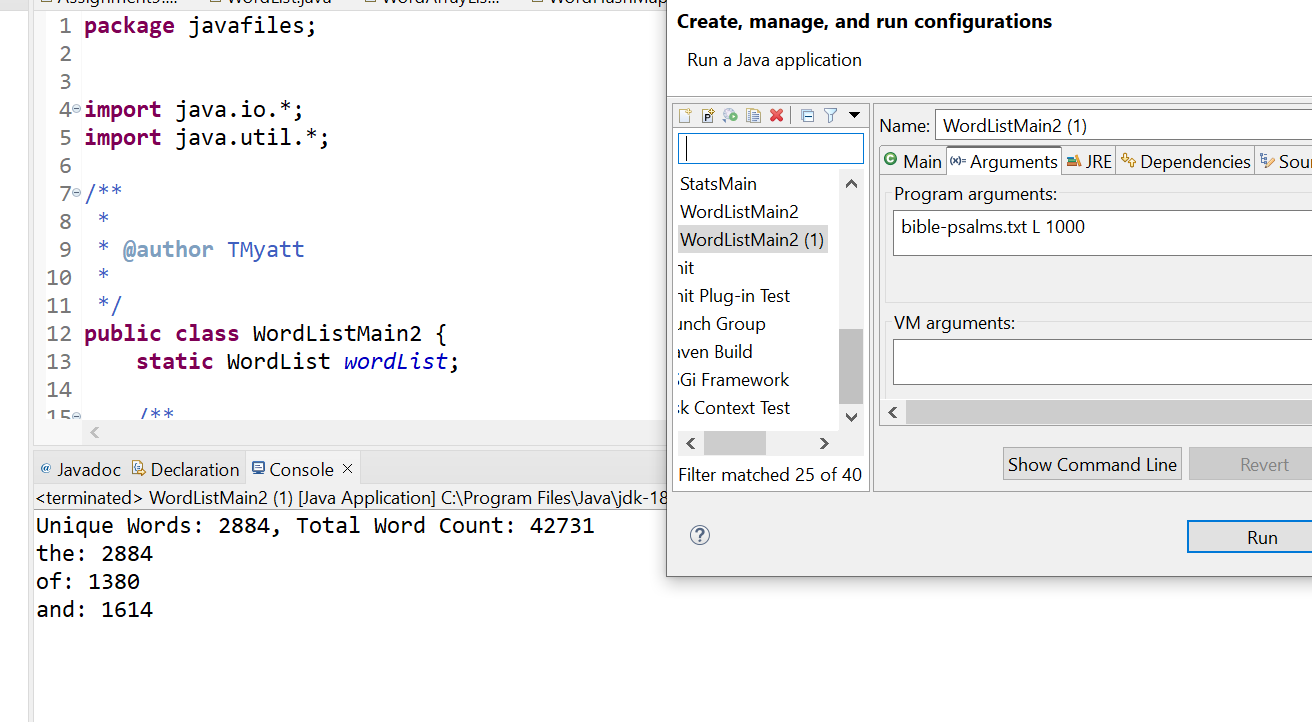
} // END catch

} // END main

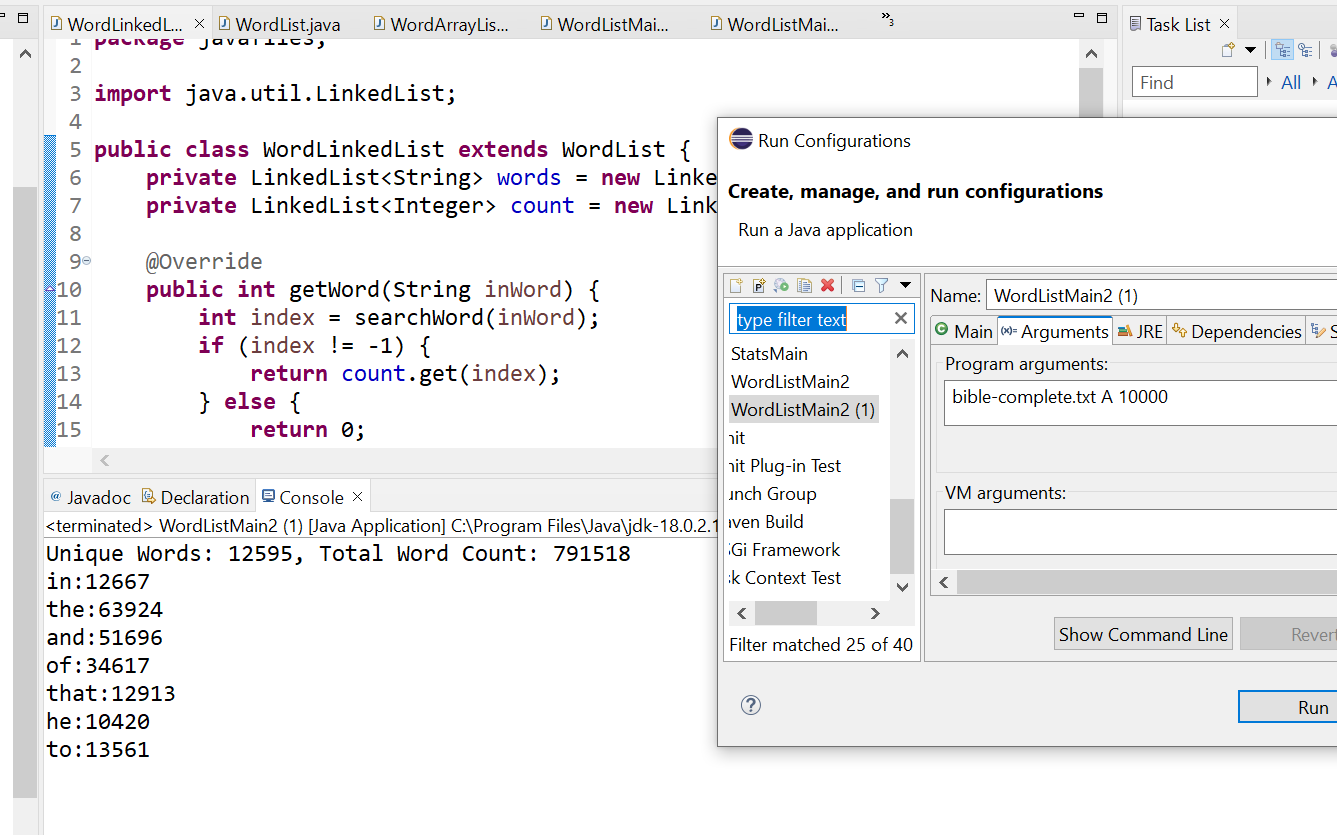
***Tests on bible-pslams.txt with 1000 limit*:**

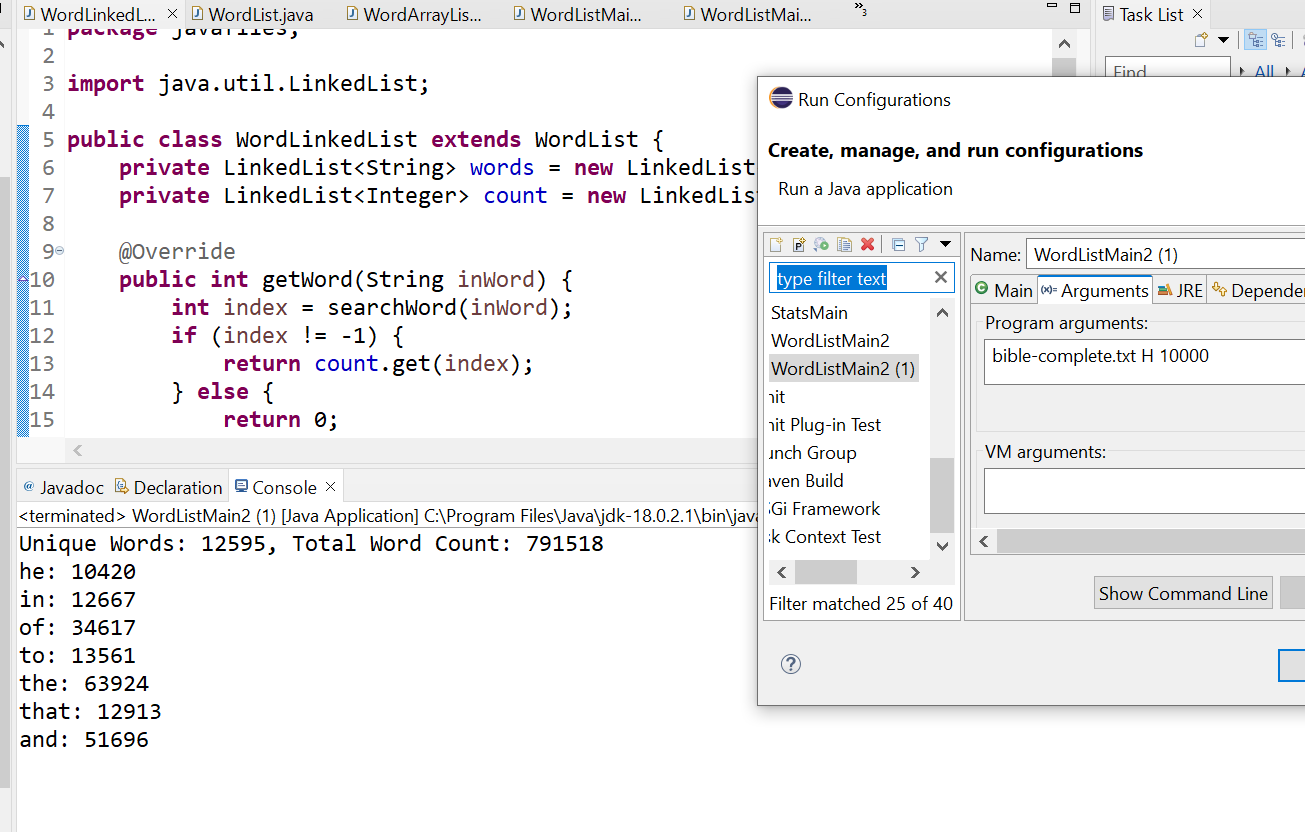
****



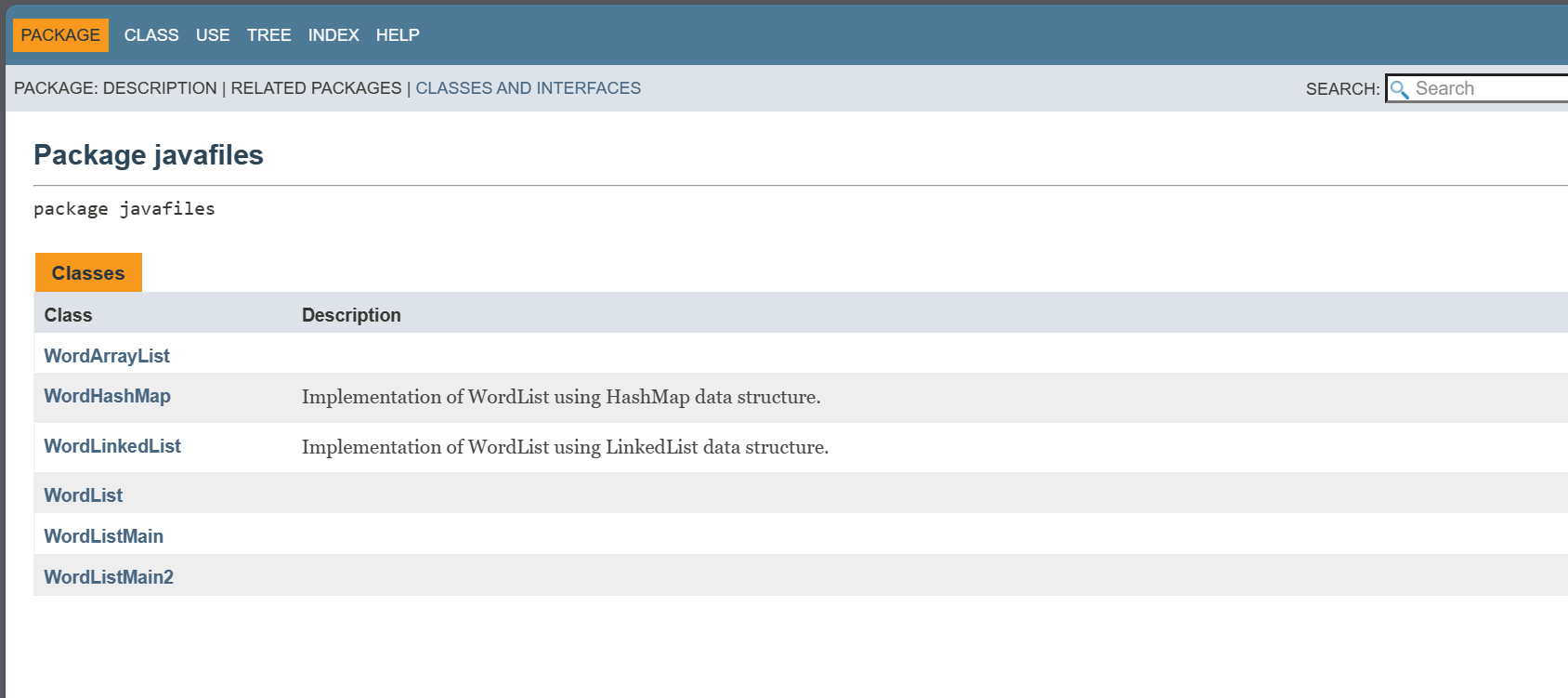
****

***Tests on bible-complete.txt*:**

****

****

***Java Docs:***

******

