```
import javax.swing.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.HashMap;
import java.util.Scanner;
import java.util.ArrayList;

public class TextAnalysis {

    // Raw Score Adjustments
    public static final long PDW_ADJUSTMENT = (long) 0.1579;
    public static final long ASL_ADJUSTMENT = (long) 0.0479;

    // Adjusted Score
    public static final long SCORE_ADJUSTMENT = (long) 3.6365;
    public static final long ADJUSTED_SCORE_CROSSOVER = (long) .05; // 5%

    // Files
    private File wordList;

    // Data
    private long average_sentence_length;
    private long difficult_word_percentage;

    // Score
    private long raw_score;
    private long adjusted_score;

    private HashMap<Character, Integer> character_map;
    private ArrayList<ArrayList<String>> wordLists;

    public TextAnalysis(File wordList) {
        setupHashMap();
       this.wordList = wordList;
       wordLists = convertWordsToList(wordList);

    }

    public static void main(String[] args) throws FileNotFoundException {
        System.out.println("Pick File to Analyze");
        File text = TextAnalysis.pickFile();

        TextAnalysis txt = new TextAnalysis(new
File("//Users//Joseph//Desktop//CSC140//WordList.txt"));

        System.out.println("\n\nFile Analysis");
        txt.analyzeFile(text);
        //txt.analyzeFile(new
File("//Users//Joseph//Desktop//CSC140//TextAnalysis//Fox.txt"));

        System.out.println(txt);
    }


    /**
     *
     * @param textFile File to analyze
     */
    public void analyzeFile(File textFile) throws FileNotFoundException {
        long word_count = (long) 0.0;
        long difficult_words = (long) 0.0;
        long sentence_count = (long) 0.0;

        Scanner textFile_scanner = new Scanner(textFile);

        String line;
        String[] words;
        String[] sentences;

        while (textFile_scanner.hasNextLine()) {
            line = textFile_scanner.nextLine();
            System.out.println("Line: " + line);

             if (!line.equals("")) {
                 sentences = line.split("[!?.:]+");
                 sentence_count += sentences.length;

                 for (String sentence : sentences) {

                     if (sentence.equals(" ") || sentence == null)
{ sentence_count--; continue; }

                     System.out.println("Sentence: " + sentence);

                     sentence = sentence.replaceAll("[!?,.:;@#$%^&*-_()]+", "");
                     sentence = sentence.replaceAll("\\b\\d+\\b", "");

                     words = sentence.split("\\s+");

                     System.out.print("Words in Sentence: ");
                     for (String word : words) { System.out.print(word + " "); }

                     int word_cnt = words.length;
                     int diff_words = countDifficultWords(words);

                     System.out.println("\nWords: " + word_cnt + ", Difficult
Words: " + diff_words);
                     difficult_words += diff_words;
                     word_count += word_cnt;

                 }
             }
        }

        average_sentence_length = word_count / sentence_count;
        difficult_word_percentage = difficult_words / word_count;
        calculateRawScore();
        calculateAdjustedScore();

    }

    /**
     * Each letter has a list of words that are placed in a larger list
     * Use the #character_map to access
     *
     * @param word_list File that contains the word list
     * @return The generated list of lists.
     */

    public ArrayList<ArrayList<String>> convertWordsToList(File word_list) {
        ArrayList<ArrayList<String>> wordLists = new
ArrayList<ArrayList<String>>();
        Scanner wordList_scanner = null;

        try {
            wordList_scanner = new Scanner(word_list);

        } catch (FileNotFoundException e) {
            e.printStackTrace();
            return wordLists;
        }

        String line;
        String[] words_in_line;

        while (wordList_scanner.hasNextLine()) {
            ArrayList<String> currentList = new ArrayList<String>();

            line = wordList_scanner.nextLine();
            if (line.equals("")) { continue; }
            currentList.add((line.charAt(0) + "").toUpperCase());

            System.out.println("Current List: "+ currentList.get(0).charAt(0));

            while(!line.equals("") && (line.charAt(0) +
"").equalsIgnoreCase((currentList.get(0).charAt(0) + "")) &&
wordList_scanner.hasNextLine()) {

                line = wordList_scanner.nextLine();
                words_in_line = line.split("\\s+");

                System.out.println("Current Line: " + line);

                for (String word : words_in_line) {
                    if (!line.equals("") && (word.charAt(0) +
"").equalsIgnoreCase(currentList.get(0).charAt(0) + "")) {
                        currentList.add(word.toUpperCase());
                        System.out.println("Added Word: " + word.toUpperCase());

                    }
                }

            }

            wordLists.add(currentList);
        }

        return wordLists;
    }

    /**
     * Counts the number of "difficult words" in an array
     * Check if the word is on a list of 3000 easy words
     *
     * @param target_words The words to check
     * @return Count of difficult words
     */

    private int countDifficultWords(String[] target_words) {
        int difficult_words = 0;

        for (String target_word : target_words) {
            if (target_word != null && !target_word.equals("") && !
target_word.equals(" ")) {
                difficult_words += isWordDifficult(target_word) ? 1 : 0; // Ternary
operator
            }
        }

        return difficult_words;

    }


    private boolean isWordDifficult(String word) {
        ArrayList<String> word_list =
wordLists.get(character_map.getOrDefault((word.charAt(0) + "").toUpperCase(), 0));

        for (String entry : word_list) {
            if (entry.contains(word)) {
                return true;
            }
        }

        return false;
    }


    /** @return The Average Sentence Length */
    public long getASL() { return average_sentence_length; }

    /** @return Percentage of Difficult Words */
    public long getPDW() { return difficult_word_percentage; }

    /** HashMap that maps the character to the index of the appropriate word list
*/

    private void setupHashMap() {
        character_map = new HashMap<>();
        character_map.put('A', 0);
        character_map.put('B', 1);
        character_map.put('C', 2);
        character_map.put('D', 3);
        character_map.put('E', 4);
        character_map.put('F', 5);
        character_map.put('G', 6);
        character_map.put('H', 7);
        character_map.put('I', 8);
        character_map.put('J', 9);
        character_map.put('K', 10);
        character_map.put('L', 11);
        character_map.put('M', 12);
        character_map.put('N', 13);
        character_map.put('O', 14);
        character_map.put('P', 15);
        character_map.put('Q', 16);
        character_map.put('R', 17);
        character_map.put('S', 18);
        character_map.put('T', 19);
        character_map.put('U', 20);
        character_map.put('V', 21);
        character_map.put('W', 22);
        character_map.put('X', 23);
        character_map.put('Y', 24);
        character_map.put('Z', 25);
    }

    /** User a JFileChooser to pick a file */

    public static File pickFile() {

        JFileChooser chooser = new JFileChooser();
        return chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION ?
chooser.getSelectedFile() : null;

        /*
        if(chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
            return chooser.getSelectedFile();

        } else {
            return null;

        }
        */
    }

    /**
     * Calculate the Raw Readability Score
     *
     * Raw Score = (0.1579 * PDW) + (0.0496 * ASL)
     *
     * @return Raw Readability Score
     */

    public void calculateRawScore() {
        raw_score = (PDW_ADJUSTMENT * difficult_word_percentage) + (ASL_ADJUSTMENT
* average_sentence_length);
    }

    public long getRawScore() { return raw_score; }

    /**
     * Calculate the Adjusted Readability Score
     *
     * The score is only adjusted if the percentage of difficult words is above the
threshold
     *
     * @return Adjusted Readability Score
     */

    public void calculateAdjustedScore() {

        adjusted_score = difficult_word_percentage >= ADJUSTED_SCORE_CROSSOVER ?
raw_score + SCORE_ADJUSTMENT : raw_score;
```

```
        /*
        if (difficult_word_percentage >= ADJUSTED_SCORE_CROSSOVER) {
            adjusted_score = raw_score + SCORE_ADJUSTMENT;
        } else {
            adjusted_score = raw_score;
        }
        */
    }

    public long getAdjustedScore() { return adjusted_score; }

    /**
     * Takes adjusted score to find the grade level
     *
     * @param adjusted_score Adjusted Readability Score
     * @return Grade Level Range
     */

    public String getGradeLevel(double adjusted_score) {
        if (adjusted_score < 5) {
            return "Grade 4 and Below - Score: " + adjusted_score;

        } else if (adjusted_score >= 5 && adjusted_score > 6) {
            return "Grade 5 or 6 - Score: " + adjusted_score;

        } else if (adjusted_score >= 6 && adjusted_score > 7) {
            return "Grade 7 or 8 - Score: " + adjusted_score;

        } else if (adjusted_score >= 7 && adjusted_score > 8) {
            return "Grade 9 or 10 - Score: " + adjusted_score;

        } else if (adjusted_score >= 8 && adjusted_score > 9) {
            return "Grade 11 or 12 - Score: " + adjusted_score;

        } else if (adjusted_score >= 9 && adjusted_score > 10) {
            return "Early College (Grades 13, 14, 15) - Score: " + adjusted_score;

        } else if (adjusted_score >= 10) {
            return "Graduate College (Grades 16+) - Score: " + adjusted_score;

        } else {
            return "Grade Not Determined - Score: " + adjusted_score;

        }
    }

    public String toString() {
        return ("Result: " + getGradeLevel(adjusted_score) + ", Raw Score: " +
raw_score + ", Adjusted Score: " + adjusted_score
                + ", PDW: " + difficult_word_percentage + ", ASL: " +
average_sentence_length);

    }

    public void reset() {
        adjusted_score = 0;
        raw_score = 0;
        difficult_word_percentage = 0;
        average_sentence_length = 0;
    }
}
```