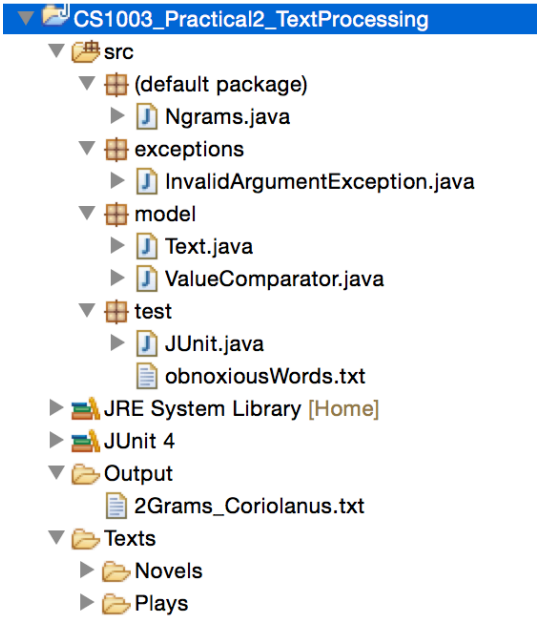
CS1003 Practical 2: Text Processing

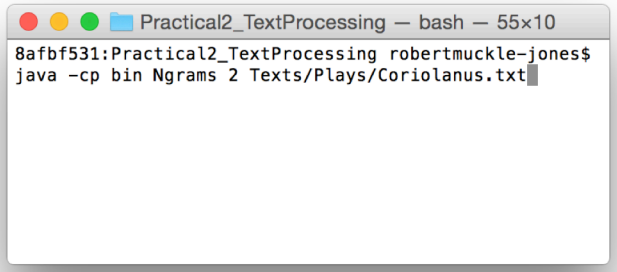
1 Overview

This report describes a Java program that reads a passage of text from a text file (e.g. a novel or a play) and generates **nGram** data for that text. An nGram is a group of letters in a word of length n
e.g. For n=3 the word 'practical' would contain seven 3Grams {'pra', 'rac', 'act', 'cti', 'tic', 'ica', 'cal'}.

2 Design



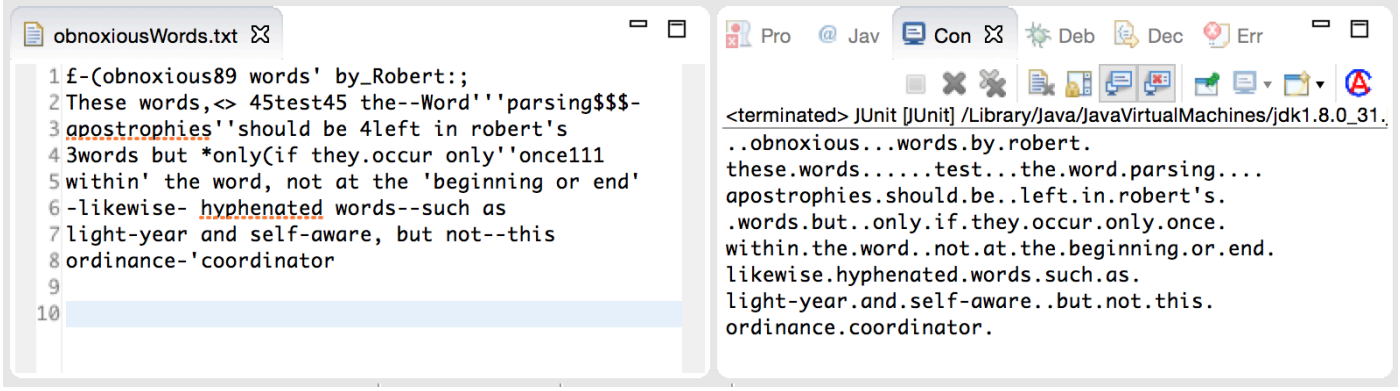
To the left you can see the package structure and the classes of the Java Program. The class **Ngrams** contains the main method and takes 2 command line arguments **[nGram num]** and **[filePath]**. The terminal window below shows an example terminal input to run Ngrams.



When NGrams is run it creates a text file in the folder 'Output' containing the nGram data for the text. In the case of the arguments passed in the example terminal window shown above a text file called "2Grams_Coriolanus.txt" is created.

The class **Text** contains the text processing methods. Most importantly it contains the methods that parse the input text into **words**, and then analyses these words to generate **nGram data** that is then stored in an appropriate data structure, whereupon a **PrintWriter** then writes this data to the output text file.

The method **Text.nGrams(int n, File document, boolean printParsing)** contains the code responsible for parsing the input text into words. It is important to ensure that the words are being parsed correctly because the subsequent text processing steps rely on correctly parsed words. Therefore, for the purposes of verification this method has a Boolean parameter, which if true, prints the word parsing to the terminal. See an example below.



As can be seen from the word parsing example, the parsing rules delimit words by any character not in the set [a-zA-Z] unless the character is an apostrophe or hyphen that occurs exactly once within a word, in which case it is considered part of the word (e.g. Robert's or light-year). The parsing implementation is achieved by the following code that incorporates **RegularExpressions**.

```
Scanner textScanner = new Scanner(new FileReader(document));

while (textScanner.hasNext()) {
    String line = textScanner.nextLine();
    String[] words = line.split("[^a-zA-Z-'']|[-'']['']"); // correctly delimit into words and remove illegal characters
    for (String word : words) { // for each word in the line
        word = word.toLowerCase();

        // the split method might have left apostrophes and hyphens at the start/end of words
        word = word.replaceAll("^\\W+", ""); // this method gets rid of any it finds at the beginning of each word
        word = word.replaceAll("\\W$+", ""); // this method gets rid of any it finds at the end or each word

        analyseWord(n, word); // this method generates nGram data and puts it in the HashMap

        if(printParsing){ // for testing purposes
            System.out.print(word + "."); // the full stop is just a visualization thing
        }
    }
    if(printParsing){
        System.out.println(); // so the word parsing output is more readable
    }

}
textScanner.close();
```

For each word that is parsed from the input text, the word is passed to a method that creates its nGrams. For each nGram that is created the nGram is stored to a **HashMap** called nGramData. The nGram (which is of type String) is itself they key to the map. Therefore nGram uniqueness is automatically enforced. The value corresponding to each nGram key in the map is the occurrence value for that nGram. This proved to by a very simple and effective way of storing all of the nGram data.

The requirement of this practical was to output the nGram data ordered by occurrence (most frequent first). A HashMap stores its data in no particular order and therefore once all of the nGram data from the text was created and stored in the HashMap this data was then put into a **TreeMap**. In the actual code there are two TreeMaps – one is sorted alphabetically by nGram(key), and the other is sorted by value(occurrence). In practice only the second TreeMap is used when the main method is called. Please refer to the source code and comments for further explanation.

3 Error Handling

In addition to the usual try/catch block to catch built in exceptions (e.g. FileNotFoundException) the mainmethod throws an InvalidArgumentException if the command line arguments aren't right. When this error is caught an informative error message is printed to the terminal reminding the user of the required arguments.

4 Testing

The Java project includes a **JUnit** test class that performs 3 tests. Each test involves reading from a file called 'obnoxiousWords.txt' located in the test package. This file includes a short passage of edge case words. You can see this passage in the figure at the bottom of page one of this report. The JUnit tests test if the program correctly calculates the most common 2Gram and 3Grams – which it does.

5 Results/Evaluation

nGrams_Works of Edgar Allan Poe									
1Gram	count	2Gram	count	3Gram	count	4Gram	count	5Gram	count
e	52967	th	13278	the	8672	tion	1330	which	653
t	39089	he	10868	and	2763	that	1049	ation	594
a	32048	in	7837	ing	2469	with	942	there	295
o	30663	er	6558	ion	1716	ther	747	would	236
i	30099	re	5891	ent	1563	hich	653	could	233
n	28551	an	5766	tio	1363	whic	653	other	208
s	25070	on	4985	hat	1362	this	650	about	199
r	23989	en	4821	her	1334	have	606	tions	193

Here is the data for the eight most common nGrams in 'Works of Edgar Allan Poe.txt' for n = 1 to 5.

The 2Gram data matches the data from the practical specification.

6 Conclusions

In this practical I learned a lot about Maps. I had never used a HashMap before, and am greatly impressed by their effectiveness. The same goes for Regular Expressions.

I can tell that using the Maps has made my code much faster than other student's solutions to this practical where Maps weren't used.

I tried to incorporate as much of the feedback from practical 1 into my code as possible. Next time I would like to learn about JavaDocs so that I can comment my methods and classes in a more organized fashion.