Full-Stack Engineer Coding Challenge

# URL Parser

## Program Documentation

Ralph

February 2019

## Overview

This program is designed for the full-stack engineer coding challenge of BrightEdge.

Given a webpage (a url), this tool is able to parse the html page, analyze its content, and output a list of key words for page classification.

## Instruction

The program is written in java and encapsulated in an executable jar file.

There are two ways to run the program.

- In console, input ***java -jar Assignment.jar <URL>***, to include the url for parsing as a passed-in value.
- Or, input ***java -jar Assignment.jar***, the program will prompt to select between sample page testing and user input url.

## Program Architecture

There are four classes defined in four java files for the program.

| File Name (src) | Java Class | Functionality |
|---|---|---|
| **TestUrl.java** | TestUrl | The test driver. Its main function is defined as the start point of the program. |
| **UrlParser.java** | UrlParser | Core class. Implement the logic to parse and analyze a webpage content. |
| **WordNode.java** | WordNode | Container for single word information. |
| **UrlValidator.java** | UrlValidator | Utility class. Do pre-processing to check the availability of a url. |

## Design Detail

### TestUrl

With user input url, the program parses the target webpage and automatically return the top k (default 20) key words.

Without user input, the program lets user to select 1 of 3 sample page for testing, or, to input a url manually and a number k for top k words, then parse that page.
In the second way, the program uses while loops to allow user to keep testing different pages without starting the program over and over.
Fault tolerance mechanism is well defined to ignore meaningless input from user. The program will call **UrlValidator** to check the availability of

inputting url before parsing every page.

## UrlParser & WordNode

The idea to get the most relevant words of a page is to parse the text on the page, extract valid words, count the frequency and select the top k words.

Noise words are pre-defined to eliminate the pronouns, prepositions and other words that appear frequently but are not helpful for identifying page topic.
The title of a page usually represents the page topic. In the program, the word extracted from a page title is assigned with a larger weight than those from the page content (the constant variable ***TITLE_BODY_RATIO***). The default value is 5, which means every word in title is counted as one appearing 5 times.

For every word, an instance of **WordNode** class is created to store its String value (field of *word*) and frequency in the page (filed of *count*).
WordNode class overrides the ***compareTo()***, ***equals(), hashCode()*** funcation so that the instance can be sorted according to its frequeney (count), from largest to smallest.

An instance of **UrlParser** is created for each page. It uses a HashMap to store the pair of word string value and its WordNode instance.
Calling the public method ***parseText()***, the program parses the page following below procedure: it applies the ***Jsoup*** library to parse the page content into single words; for every incoming word, the program checks if it

is noise, get its WordNode instance from HashMap, then update its frequency in instance.
By calling the public method ***printTopK()***, the program moves all WordNode into a TreeSet to do the sorting. Then, it applies an iterator to traverse the BST, from the most frequent word to the least one, till getting all top k words.

There are several considerations for this design.

1. The methods in UrlParser are defined as instance methods, so that it allows parsing multiple page concurrently, with multiple instances.
2. The ***parseText()*** and ***printTopK()*** are defined as public methods and they can be called separately. For example, the program can parse multiple pages, then, print out the top k words on multiple pages. Fault checking mechanism is defined to avoid illegal input and situation.
3. TreeSet is applied to sort the word according to frequency. It costs O(nlogn) time to build up the BST with all words, and allows to get top k nodes without removing them (compared to *poll()* in PriorityQueue).

## UrlValidator

This class is defined as a utility to pre-check the availability of an input url, so all fields and method are defined as static.
Upon running, it tries to connect to the url for certain times, as pre-defined in constant variable ***RETRY_LIMIT*** (default 5). A url will pass the checking if it gets a response with state code of 200.