Coursework 1

Important Dates

Handed out: 2 - Feb -2018 **Handed in**: 24 - Feb -2018 09:00 am

About this coursework

- This coursework counts as 13.33% of your final mark (33% of the coursework marks).
- This coursework is an individual piece of work. Please read guidelines on plagiarism in the studyguide and course documentation.
- This coursework requires knowledge about Java programming and Multi-Threading.
- Please scan or take a picture of your signed coversheet and upload it together with your submission.

Context

Word frequency analysis is a technique for text mining and classification. It has been used widely in research to gather information from long texts, in order to determine patterns and trends. Once the the frequencies have been obtained it can be presented graphically as shown below.

Figure 1: Word Cloud: The size represents higher frequency

Goal

The aim of this coursework is to implement a multi-threaded application to analyse the word frequency on a given website.

Task 1: 50 Marks

You need to implement the **WebMiner.java** and **MinerManager.java** classes using the threaded approach of your preference. In essence your application should:

- Given a root web url (e.g. https://bbc.co.uk) and a list of n keywords, your application should extract the hyperlinks and contents to:
  - Carry out the text mining of the contents obtaining word frequency for every keyword
  - Repeat the process for every hyperlink obtained
  - Calculate: the total number of web pages visited, the number occurrences of every key word per every visited web site, the total number of occurrences of every key word for all the visited pages and the number of employed web miners.
- At the end, your application should show the gathered statistics in a simple text format.

See the Appendix at the end.
Note: Use appropriate thread-safe type when necessary. You are allowed to use built-in java collection classes (e.g.Vector, ArrayBlockingQueue and ConcurrentHashMap etc) - Extra bonus will be given to those who implement/use their own version of thread-safe list, map or queue)

Task 2: 30 Marks

You need to implement the following functionality, either complementing the classes previously developed or with new classes:

- The MinerManager thread must be able to indicate to the working WebMiners to stop mining when any of these conditions are met:
  - There are no more pages to process
  - A maximum number of pages have been visited
  - A maximum number of keywords have been found among the visited pages
  - A time out has been reached
- The MinerManager thread must be able to limit the maximum number of WebMiners working simultaneously

Task 3: 20 Marks

1. Briefly explain two characteristics of your Multi-Thread implementation that make it better than a single-threaded counterpart. [5 Marks]
2. Pick, as maximum, the 5 most important active and passive objects of your implementation and draw diagram showing the read/write relations. [5 marks]
3. Briefly explain how did you avoid, in your solution, the interference of the threads when working with shared resources. [5 Marks]
4. Please describe how your application behaves after the MineManager requests the WebMiners to stop mining and explain the reasons of this behaviour. [5 Marks]

Note: If you did not manage to complete your implementation, answer these questions according to what you have learned during this course and what would you expect to happen in the application.

Resources

The following Java Classes are provided:

Class	Purpose	Comments
WebMiner.java	This class is generate objects that behave as Threads, exploring and mining the web content.	You need to complete.
MineManager.java	This class is to generate a (Thread) manager for the miners. This class provides methods for controlling, monitoring and summarising the job done by miners.	You need to complete.
Utils.java	This class provides static methods that you will find useful to explore web sites, obtain their contents in plain text format, striping tags and counting frequency of words.	No need to modify.
DemoUtils.java	This class gives you an example on how to use the methods from the Utils.java class.	No need to modify.

- See Java Docs for information of parameters of each class' methods
- If you consider it necessary, it is allowed to make changes in any on the classes provided (or create new classes) as long as the Multi-threading behaviour is guaranteed.

Instructions

- Create a java package uk.ac.le.cs.CO3090.cw1,import all Java files to the package.
- Download jsoup-1.8.3.jar and add to the Java Build Path (Eclipse -> right-clicking on the Project -> Build Path -> Configure Build Path. Under Libraries tab, click Add External JARs). You can also use Gradle or Maven to manage jar libraries dependencies.
- You need to submit your solution through BlackBoard. A complete submission includes:
  - Your signed cover (plagiarism) sheet
  - A pdf with the answers to question 3
  - A package with all the source code (.java files) that is required to compile and run your solution.

Appendix

Breadth first search (BFS) and depth first search. (DFS)

Breadth first search (BFS) and **depth first search (DFS)** are two of the most important search algorithms.

Input:

```
rootURL: String   // root/base URL where the mining starts
Q:Queue           //a FIFO queue for storing the URLs to be visited
visited:List      //contains a list of URLs already visited
results:Map<String, Integer>
                  // used for storing occurrences (count) of each keyword
```

Pseudocode for **BFS** search:

```
Main thread:

enqueue rootURL to Q

while Q is not empty then
   start a new WebMiner thread t and invoke t.mine(URL)
```

```
WebMiner threads:

procedure mine(URL)

   dequeue a URL from Q
   add URL to visited
   count words, update results
       for each hyperlink on the page
             enqueue hyperlink onto Q
```

Note: All `WebMiner` threads (except the first thread) will obtain the URL from a shared queue.

DFS: using a Stack instead of a Queue would turn the BFS algorithm above into a DFS search. Alternatively, you could use a recursive implementation of DFS:

Pseudocode for **DFS** search:

```
Main thread:

start a new WebMiner thread;
```

```
WebMiner threads:

procedure mine(URL)
     add URL to visited
     count words, update results
     for each hyperlink on URL
             start a new WebMiner thread t and invoke t.mine(hyperlink)
```