GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computer Science & Applied Physics

Data Structures & Algorithms
ASSIGNMENT DESCRIPTION & SCHEDULE

A Word Cloud Generator

Note: This assignment will constitute 50% of the total marks for this module.

Overview
Word-clouds are a mechanism for creating a visual representation of text and are used to display a visual summary of the most prominent words used on a web page, a news forum or a social media web site. A word-cloud is comprised of a set of tags, with each tag representing a single word. The prominence of a word is typically estimated from its occurrence frequency in a text and is visualised using a large font size or different font colour.

You are required to develop a Java application that can parse a **file or a URL** to generate a PNG (portable network graphics) file with a word-cloud displaying the most prominent words in decreasing font size, style and colour. Note that your application should ignore frequently occurring words and HTML tags, in the case of a word-cloud generated from a URL. A list of common words is provided in the file ***stopwords.txt***.

Minimum Requirements
The application must contain the following features:

- A ***command-line menu*** that prompts that present the user with a suite of choices, including:
  - The name of the file or URL. Do not hardcode file names, paths, URL or any environmental variables into your application. Doing so is a cardinal sin in computer programming. Instead, ask the user to enter these variables as input parameters from the menu.
  - The maximum number of words to display.
  - The file name of the word-cloud image to save.
- A ***parser*** that reads the file or URL line-by-line, extracts each word and adds it to a frequency table.

- The ***frequency table*** can be implemented as a *Map<String, Integer>*, where each word is stored as a String key and the frequency as an integer. Note that that a far more space efficient map can be used with very little additional effort…
- Only words that are not in the file **ignorewords.txt** should be added to the map. You are free to add extra words to the file *ignorewords.txt* if warranted. As every word encountered by the parser must be checked against the words in *ignorewords.txt*, this operation must be **very** fast. You can assume that the file is available in the current directory and should refer to it as "./ignorewords.txt".
- You are free to ***implement the drawing of the words in the word cloud*** any way you wish. An example of how to create a PNG from text is provided at the end of this document.
- You must **comment each method** in your application stating its running time (in Big-O notation) and your rationale for its estimation.

Please note that extra marks will be given for the appropriate application of the ideas and principles we study on this course. The emphasis should be on speed (low time complexity) and a minimal amount of RAM consumption (low space complexity).

Deployment and Submission

- ***The project must be submitted by midnight on Sunday 21st April 2019*** as a Zip archive ***(not a rar or WinRar file)*** using the Moodle upload utility. You can find the area to upload the project under the *"A Word Cloud Generator - (50%) Assignment Upload"* heading in Moodle.
- You must use the ***package name*** **ie.gmit.sw** for the assignment. There is also a set of small, medium and large text files, under the "Sample Text Files for Assignment" link that you can use to test your application.
- ***Do not submit any text files*** with your application or any extra APIs (Java libraries).
- The ***name of the Zip archive*** should be *{id}*.zip where *{id}* is your student number.
- The Zip archive should have the following ***structure*** (do NOT submit the assignment as an Eclipse project):

Name	Description
src	A directory that contains the packaged source code for your application. You must package your application with the namespace **ie.gmit.sw**. Your code should be well commented and explain the space and time complexity of its methods.
README.txt	A text file outlining the main features of your application.
ignorewords.txt	Only include this file if you have made alterations to the one provided. You can ***assume that the file is available in the current directory*** and should refer to it as "./ignorewords.txt".
wordcloud.jar	A Java Application Archive containing the classes in your application. Use the following syntax to create a JAR from a command prompt: **jar –cf wordcloud.jar ie/gmit/sw/*.class** Your application should be launched using the following syntax from a command prompt: **java –cp ./wordcloud.jar ie.gmit.sw.Runner** **Note:** this requires that the ***main()*** method is defined inside a class called *Runner.java*.

Scoring Rubric

The following marking scheme will be used to score the project:

Component	Marks (100)	Description
Submission	5	Zip archive correctly named and structured
Compiles	5	All or nothing. The Java source code compiles without errors and runs from the JAR file.
README	5	The README file clearly describes the application, it's features, is free from spelling and grammatical errors.
Jar	5	All or nothing. The JAR file is correctly named and allows the application to be launched as defined in the spec.
UI	13	The user interface allows a file, URL and maximum number of words to be specified and has options for output names and quit.
File/URL Parser	23	The parser works correctly for a file or URL and with a reasonable space and time complexity.
Output	24	The application can draw and output a PNG to file.
Big-O	10	All methods and key steps in the application have been fully commented with Big-O running times.
Extras	10	Any additional functionality that has been documented in the README. The extras must be relevant to data structures and algorithms.

Creating a PNG Image from Text

The Java 2D API provides a rich set of classes for manipulating images. The capabilities of the *BufferedImage*, *Graphics* and *ImageIO* classes are amply sufficient for this project. We can create a *BufferedImage* of a given size and use its associated *Graphics* object to draw text onto an image. The image can then be converted to a PNG and saved using the *ImageIO* class. For the more intrepid and discerning programmers amongst you, the *Graphics* object can be cast to a *Graphics2D* type, provide an even richer graphics environment that includes lighting, shadowing, ghosting and other effects.

```
import java.awt.*;
import java.awt.image.*;
import javax.imageio.*;
import java.io.*;

public class ReallySimpleWordCloud{
   public static void main(String args[]) throws Exception{
      Font font = new Font(Font.SANS_SERIF, Font.BOLD, 62);
      BufferedImage image = new BufferedImage(600, 300, BufferedImage.TYPE_4BYTE_ABGR);
      Graphics graphics = image.getGraphics();
      graphics.setColor(Color.red);
      graphics.setFont(font);
      graphics.drawString("Data Structures", 0, 100);

      font = new Font(Font.SANS_SERIF, Font.ITALIC, 42);
      graphics.setFont(font);
      graphics.setColor(Color.yellow);
      graphics.drawString("and Algorithms", 10, 150);

      font = new Font(Font.MONOSPACED, Font.PLAIN, 22);
      graphics.setFont(font);
      graphics.setColor(Color.blue);
      graphics.drawString("2019 Assignment", 40, 180);

      graphics.dispose();
      ImageIO.write(image, "png", new File("image.png"));
   }
}
```