GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computer Science & Applied Physics

H.Dip in Software Development

Advanced Object Oriented Software Development
Main Assignment (50%)

Generating a Word Cloud from Text

1. 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*** on Moodle.

2. Minimum Requirements

- Use the package name ***ie.gmit.dip***. The application must be deployed and runnable using the specification in Section 3.
- Create a console-based ***menu-driven UI*** that presents 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.

  Any other input parameters should be input through the menu-driven UI.

- 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.
- Provide a **UML** diagram of your design and fully **JavaDoc** your code.

Note that the whole point of this assignment is for you to demonstrate an understanding of the principles of object-oriented design and data structures by using abstraction, encapsulation, composition, inheritance and polymorphism WELL throughout the application. You are free to asset-strip any online resources for text, images and functionality provided that you modify any code used and cite the source both in the README and inline as a code comment above the relevant section of the programme. You are not free to re-use whole components and will only be given marks for work that you have undertaken yourself. Please pay particular attention to how your application must be packaged and submitted and the scoring rubric provided. Marks will only be awarded for features described in the scoring rubric.

3. Deployment and Delivery

The project must be submitted by midnight on 6th January 2022. Before submitting the assignment, you should review and test it from a command prompt on ***a different computer*** to the one that you used to program the project.

- The project must be submitted as a Zip archive ***(not a 7z, rar or WinRar file)*** using the Moodle upload utility. You can find the area to upload the project under the "*Generating a Word Cloud from Text (50%) Assignment Upload*" heading of Moodle. *Do not add comments to the Moodle assignment upload form.*
- The name of the Zip archive should be *<id>*.zip where *<id>* is your GMIT student number.
- The Zip archive should have the following structure (do NOT submit the assignment as an Eclipse project):

Marks	Category
wcloud.jar	A Java **archive** containing your API and runner class with a main() method. You can create the JAR file using Ant or with the following command from inside the "bin" folder of the Eclipse project: **jar –cf wcloud.jar *** The application should be executable from a command line as follows: **java -cp ./wcloud.jar ie.gmit.dip.Runner**
src	A directory that contains the packaged **source code** for your application.
README.txt	A text file detailing the main **features** of your application. Marks will only be given for features that are described in the README.
design.png	A UML **class diagram** of your API design. The UML diagram should only show the relationships between the key classes in your design. Do not show private methods or attributes in your class diagram. You can create high quality UML diagrams online at **www.draw.io**.

docs	A directory containing the **JavaDocs** for your application. You can generate JavaDocs using Ant or with the following command from inside the "**src**" folder of the Eclipse project: **javadoc -d [*path to javadoc destination directory*] ie.gmit.dip** Make sure that you read the JavaDoc tutorial provided on Moodle and comment your source code correctly using the JavaDoc standard.

4. Marking Scheme

Marks for the project will be applied using the following criteria:

Element	Marks	Description
Structure	8	The packaging and deployment correct. All JAR, module, package and runner-class names are correct.
README	8	All features and their design rationale are fully documented. The README should clearly document where and why any design patterns have been used.
UML	8	Class diagram correctly shows all the important structures and relationships between types.
JavaDocs	8	All classes are fully commented using the **JavaDoc** standard and generated docs available in the *docs* directory.
Robustness	38	The application executes perfectly, without any manual intervention, using the specified execution requirements.
Cohesion	10	There is very high cohesion between packages, types and methods.
Coupling	10	The API design promotes loose coupling at every level.
Extras	10	Only relevant extras that have been fully documented in the README.

You should treat this assignment as a project specification. Each of the elements above will be scored using the following criteria:

- 0–30% **Fail:** Not delivering on basic expectations
- 40-59% **Mediocre:** Meets basic expectations
- 60–79% **Good:** Exceeds expectations.
- 80-89% **Excellent:** Demonstrates independent learning.
- 90-100% **Exemplary:** Good enough to be used as a teaching aid

5. 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("H.Dip AOOSD Assignment", 40, 180);

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