Department of Computer Science & Applied Physics

H.Dip. in Science (Software Development)
Object Oriented Programming
2024 ASSIGNMENT DESCRIPTION & SCHEDULE

Simplifying Text with Word Embeddings and Virtual Threads

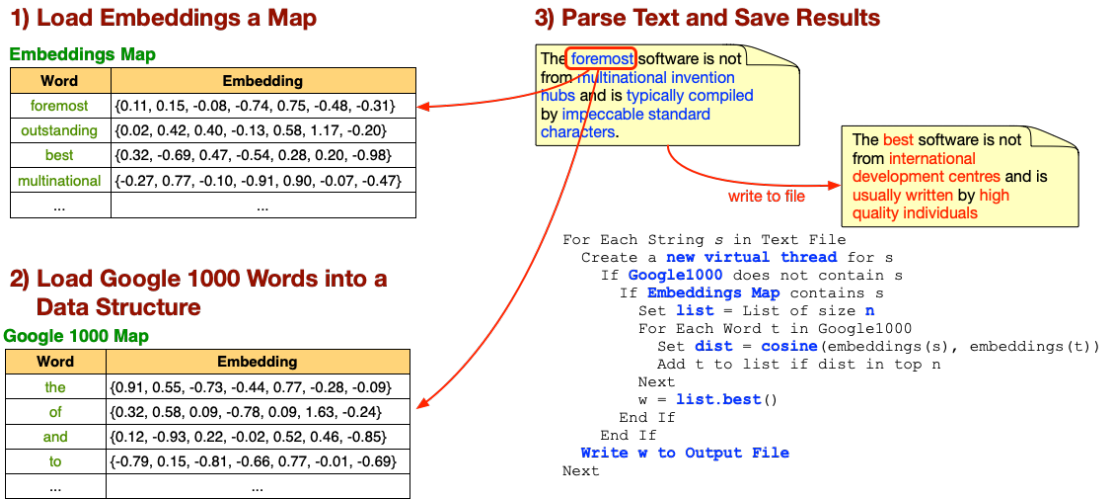
***Note*: This assignment is worth 50% of the total marks for this module.**

1. Overview

You are required to design and develop a text simplifier API and application UI in Java that converts the words in a text file to the 1,000 most common words in English.

> "*Never use a long word when a short one will do. Never use a foreign phrase, a scientific word, or a jargon word if you can think of an everyday English equivalent.*" — George Orwell.

The application should allow a user to enter the location of a text file and dynamically swap every word for its synonym from the Google list of 1,000 most common words in English.



The application should process a specified target text file word-by-word and dynamically swap every word for its synonym from the Google-1000 list. The similarity between words should be computed from the reduced set of 59,602 GloVE word embeddings provided. As shown in the diagram above, this will require loading and processing all word embeddings into a map data structure, generating a second map that relates each Google-1000 word with its embedding, and then searching / swapping each word in the target text with its Google-1000 equivalent based on the highest similarity score. The application must use virtual threads to concurrently process each word in the embeddings, Google-1000 and target text files. When the target text file has been processed, the results should be saved to specified file.

2. Minimum Requirements

- You must use the package name ***ie.atu.sw***. The application must be deployed and runnable using the specification in Section 3. You should use the suite of stubs provided on the VLE.

- **Provide a simple command-line user interface** that enables a user to specify the following:
    1. A path and name for the word embeddings file to use.
    2. A path and name for the text file to simplify.
    3. A path and name for the file to output.

  You can include as many other menu items as you wish and will be given marks for relevant functionality added.

- The application should use ***virtual threads*** to process the word embeddings file, the Google 1,000 list and the text file to simplify. You are encouraged to use ***structured concurrency*** where appropriate. You should give careful consideration to using the suite of data structures in the **java.util.concurrent** package.

- **Search** the Google 1000 word embeddings, using one or more vector comparison algorithms, for the highest scoring similar word to each target word.

- **Output** the simplified text to a file.

- You must ***comment each method*** in your application stating its running time (in Big-O notation) and your rationale for its estimation.

- **Abstract and encapsulate** your design to promote high cohesion and loose coupling between the different types used in your application.

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

- Provide a **README** file in PDF format detailing the main features of your application in no more than 300 words. Marks will only be given for features that are described in the README.

Copying Code, Plagiarism and Generative AI

You are free to asset-strip any online resources for 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.

Generative AI

You can use generative AI to create components, provided that these are then **significantly modified / adapted** and the **original generated code** and the **prompt used** for its creation are **included** at the end of the README.

3. Deployment and Delivery

The project must be submitted by 5pm on 6th January 2025. 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.

Do NOT submit the assignment as an Eclipse project or submit any text files or other resources. If you do, you will lose marks. You will also lose marks if you hard-code any environmental variables in your project like system paths and file names.

- The project must be submitted as a **Zip archive** *(not a 7z, rar or WinRar file)* using the VLE upload utility. You can find the area to upload the project under the "*Simplifying Text with Word Embeddings and Virtual Threads (50%) Assignment Upload*" heading of the VLE. *Do not add comments to the VLE assignment upload form.*
- The Zip archive should be **named *<id>*.zip** where <*id*> is your ATU student number.
- The Zip archive **should have the following structure**:

Marks	Category
oop.jar	A Java **archive** containing your API and runner class with a *main()* method. Use the following command from inside the "bin" folder of the Eclipse project: **jar -cf oop.jar *** The application should be executable from a command line as follows: **java -cp ./oop.jar ie.atu.sw.Runner**
src	A directory that contains the packaged **source code** for your application.
README.pdf	A **PDF file** detailing the main features of your application in **no more than 300 words**. Marks will only be given for features that are described in the README.
design.png	A UML **class diagram** of your API design showing 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.atu.sw** Make sure that you read the JavaDoc tutorial provided on the VLE and comment your source code correctly using the JavaDoc standard.

4. Marking Scheme

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

Component	Marks	Description
Robustness	40	The application deploys and executes correctly from the JAR as specified.
Design	30	High cohesion and loose coupling applied throughout the design.
Documentation	25	Comments and supporting documentation, including a UML diagram.
Extras	5	Only relevant extras that have been fully documented in the README.

Range	Expectation
0–39%	***Not delivering*** on basic expectations. The submission does not meet the minimum requirements.
40-60%	Meets ***basic*** expectations. The minimum requirements are met in whole or in part.
61–74%	Tending to exceed expectations. A ***high-quality*** assignment with additional functionality added.
75-89%	Exceeding expectations and ***very high-quality***. Any mark over 70% requires evidence of ***independent learning*** and cross-fertilization of ideas, i.e. your project must implement a set of features using advanced concepts not covered in depth during the module.
90-100%	***Exemplary***. Your assignment can be used as a teaching aid with little or no editing.