Massey University

159.251 - Software Design and Construction

Assignment 1

Deadline and Late Submission Penalties

You must submit your final work using the stream submission system no later than **19 October 2025 at 11.59 pm**. The penalty is 20% deducted from the total possible mark for every day delay in submission (one day late – out of 80%, two days late then out of 60% … etc.).

You are expected to manage your source code and other deliverables using version control. "*The Cat Ate My Source Code*" is not a valid excuse for a late submission.

Contribution to Final Grade of the Course: **22%**

Please read carefully, as there are many parts that you should be aware of.

Overview

You are to work in ***self-selected pairs*** (i.e., select your teammate) to create the program defined below. You will need to use ***git*** to manage source code contribution and integration between the two members. All project issues and changes should be tracked using an ***issue tracker*** (set up within your repository).

Note: Both group members will receive the same mark unless it is clear that the work is predominantly that of a single person. These will be sorted out on a case-by-case basis. The partition of the work is entirely up to you and your project partner.

Part of this assignment is to become familiar with using ***git*** for version control. You must use GitHub for this assignment, and the ***repository must be private***.

IMPORTANT: use the following conventional name for your repository: **251-Assignment1-2025-FirstName1-FirstName2**. For example, if the first student is Sarah and the second student is Li, then the repository name should be (**251-Assignment1-2025-Sarah-Li**).

After submitting your assignment on Stream, you must send an invitation to your GitHub repository to one of our markers. **After the submission is completed,** one of the markers will contact you to request access to your repository.

Tools Required

- **IDE independent:** You may develop this in any IDE or code editor you like! The tools included here are available for major IDEs and Maven dependencies.
- Git and GitHub(for version control)
- GitHub Issue Tracker (for log changes and bugs)
- Maven (for dependency management and process automation)
- A code metrics tool
- Docker for packaging
- PMD (for code quality check)
- other libraries to perform the different functionalities (you are free to search and use any library that will help in implementing any of the functionalities).

Tasks

1) **Developing** a ***text editor*** program using Java – see details below.
2) **Source code and version control:**
    a) create and maintain a *git* repository on your local machine for your source code and on a remote repository to provide a central server accessible to both members and also accessible for marking.
    b) keep an audit trail of commits in the *git* repository. **These will inform part of the marking.**
    c) make sure that you ***actively*** (almost daily) use git features such as ***branching*** and ***merging*** (not only on the last day before submission!).
    d) write your configurations in YAML file. You must submit at least one configuration file that works with the text editor. This could be a file with the default parameters for the text editor, such as the default text format or default font colour.
3) **Log changes and bugs**: keep track of changes and issues – use an ***issue tracker*** as part of your version control. Please use the same issue tracker that is provided by your Git project hosting site (i.e., the "issues" feature in GitHub). This has to be actively used!
4) **Automation**: automate your process so it is easier to load files and generate reports. Use ***Maven*** to declare all dependencies. If you are using any external libraries, do not include any jar files with your submission but add them as Maven dependencies! Also, don't try to change your Maven files into a different format. The pom file should be an XML file.

    Build those Maven build tasks in your Continuous Integration (CI) pipeline using GitHub Actions. Make sure that the configuration (YAML) file is correctly added.
5) **Readability**: ensure you write clean code, *correctly handle exceptions*, and add comments to explain your code. Make your code "human-readable!".
6) **Quality**: check the quality of your code and outputs

a) use code quality tools to report metrics data of your program. The metrics report generated from your code should be submitted. The process should be automated and included in your ***Maven*** dependencies (see Section 2 below).

b) (Bonus) Write unit tests using JUnit to test (at least) the following functionalities: **open**, **save** and **search** (see details below).

7) Deployment: package your project and deploy it into a Docket container (a Windows or Linux container). This must allow the markers to run the Maven project from the Docker container (without the need to install or configure the environment). Make sure to include your Docker file with your project, and to include details on how to load and run your container in the README file.

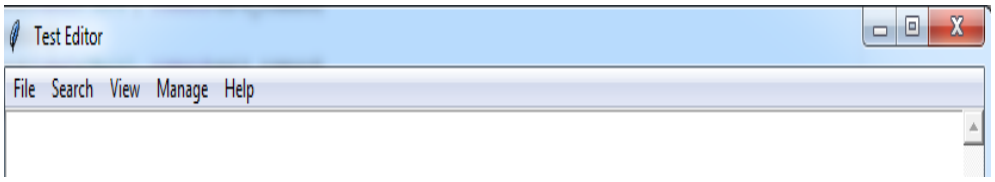
Make your own text editor!

1. The Text Editor

Your program is basically a standard text editor (or text processor) – something similar to Notepad, Atom, or Geany. The editor should allow you to write text on it using standard text encoding formats (i.e., ASCII/UTF-8). You should develop this program in **Java.** Note: a standard text file (*mostly*) does not need any additional ***metadata*** files to assist the reader in interpretation,

The main functionalities of the text editor are:

- Full GUI access to the application
- Create a menu of options at the top of the editor, similar to the following



The menu should (at least) include the following sub-menus: File, Search, View, and Help

- Implement the following functionalities:
  - **New** function: to create a new (fresh) window.
  - **Open** function: to read other text files (just **standard .txt** files). This should allow users to navigate the file system to select\search for a file.
  - The ability to **read OpenDocument Text (.odt)** files. This is part of the **Open** function.
  - **Save** function: save text output into **.txt** file format. This should allow users to navigate the file system to save the file in a selected drive/location.
  - **Search:** search for text within the screen (this will be tested based on a single word)
  - **Exit:** to quit the program – close all windows.
  - **Select text, Copy, Paste, and Cut (SCPC)** capabilities.
  - **Time and Date (T&D):** retrieve the current time and data from the OS and place it at the top of the editor's page.
  - **About:** display the names of both team members and a brief message in a popup message box.
  - **Print** function: allow your editor to print text by connecting it to the local printer in your machine (similar to any other text editor you have used).

Harder functions

- ability to read source code files such as .java, .py, .cpp or similar. ***different syntax should be shown in different colours***. For example:

```java
import java.lang.*;
/**
 * @author atahir
 * @version 1.1.1
 */
import java.util.Random;
public class foo extends bar
{
    public void act()
    {
            Random random = new Random();
            int barVal = random.nextInt(30) - 15;
            System.out.println(barVal)
    }
}
```

- reading and displaying other file formats beyond txt and source code files: mainly, Rich Text Format (**RTF**) and OpenDocument Text (**ODT)** format. Hint: you can use an external library to do so.
- include a **PDF** conversion function in your editor, so the file can be saved in PDF format (for standard text files). Use an external library for this, such as Apache PDFBox.

Note: There is no specific requirement regarding which GUI library you should use, but try to make your program as cool as possible!

2. Code Quality and Management

Once development is done, you must report metrics data using a metrics tool. You may use a software metrics tool (**see some examples above)**. Code quality report from PMD should also be submitted with your assignment.

a) generate a metrics data report from any software metrics tool (see below for the specific metrics) and add the report file (.txt or html) to $project$/reports/metrics

b) create a new ***maven*** goal called "pmd" that should generate a metrics report using **PMD** (see below for the specific metrics) and add the report files to $project$/reports/pmd.

- **Code Size (per class):** Lines of Code (LOC) and Number of Methods (NOM)
- **Code Complexity:** Cyclomatic Complexity and code coupling metrics (Coupling Between Objects (CBO) OR Efferent Coupling).
- **Code Quality Report from PMD**. Use *only* Java Basic rules such as *Naming Convention* for classes and variables (extract the full report and include it with your submission).

Submitting your assignment

Submission must be completed on Stream using the Assignment submission site.

Share your program on your ***private*** GitHub repository with us by sending a share invitation to the user. This is to track commits on your Git repository.

Include a README file in the top level of the project

The *Readme.md* is a file with a Markdown syntax (this should be correctly formatted as a markdown) that contains:

1. the names & IDs of BOTH MEMBERS of the group.
2. clear instructions on how to run your program and if there are any other directories, what they contain.
3. for each student, a couple of the most significant git ***commit IDs*** show the work of each individual member of the group.
4. a link to your private GitHub repository. A marker will contact you after submission to request access to the repository. Note: there will be a penalty if the repository is found to be public.

Who submits what?

Only one member of a group should submit a complete project, the other just submit the *Readme.md* file:

- **member A**: submit (through Stream) a single compressed (e.g., zip or tar) file that contains the assignment (Maven project files and other configuration filess without the /target directory)
    - **name the compressed file with both members' FirstName_LastName and ID numbers** (e.g. Xiaofeng_Liu-87878787-Susan_Jones-01010101.zip)
- **member B**: submit just the README.md file containing your name and that of the partner who is submitting the zip/tar file. This is so Stream knows that you've submitted something.

Markdown Quick guide
https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

Assessment

Your assessment will be based on the following criteria:

Criteria	Mark
Correct implementation of the **text editor** main window (which should also include a main menu)	1
Correct implementation of the following functions: **New**, **Open**, **Exit**, **T&D**	2
Correct implementation of the **Select text, Copy, Paste and Cut (SCPC)** functions	1
Correct implementation of the following functions: **Save**, **Search** and **Print**.	1.5
Correct implementation of the **PDF conversion** functionality	1
Correct implementation of the following functions: **Open (read) .RTF and .ODT files**	2
Correct implementations of the **read source code files** (.java, .py and .js)**.** Different syntax should be shown in **different colours.**	2
Appropriate use of git **FROM THE START OF DEVELOPMENT**	3
Appropriate use of issue tracking to track changes/issues **FROM THE START OF DEVELOPMENT.**	2
Correct use of ***Maven*** with all external dependencies correctly added and the project can compile and run without needing the IDE.	1.5
Correct deployment into Docker container (upload to load and run correctly)	1.5
Correct use of configuration files (in YAML)	1
Correct use and implementation of the CI using GitHub Actions	0.5
Check code quality and include reports of the size, complexity, and other quality metrics. PMD is also added as a Maven goal	1
Overall code quality, including exception handling and comments to explain the code.	1
(extra/bonus) High-quality unit tests added	+1
Total	**22 (max)**