Massey University

159.251 - Software Design and Construction

Assignment 1 - 2022

Deadlines and penalties

You must submit your final work using the stream submission system no later than **28 August at 11.59pm**. The penalty is 10% deducted from the total possible mark for every day delay in submission (one day late – out of 90%, two days late then out of 80% … etc.).

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

Contribution to Final Grade of the Course: 22%

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, using ***git*** to manage source code contribution and integration between the two developers. All project issues and changes should be tracked using ***issue tracker software***.

Note: Both members of the group 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 will need to use GitHub for this assignment, and the ***repository must be private***.

IMPORTANT: use the following conventional name for your repository: **251-Assignment1-2022-FirstName1-FirstName2**. For example, if the first student is Sarah and second student is Li, then the repository name should be (**251-Assignment1-2022-Sarah-Li**).

You will need to send an invitation to your repository to one of our markers after the submission. One of the markers will be in touch with you after the submission is completed to request access to your repository .

This is important and will be part of your assessment.

Tools Required

- **IDE independent:** you may develop this in any IDE or code editor you like! Tools included here are available for major IDEs, and also as Maven dependencies. Any Java IDE (Eclipse or IntelliJ or any other of your choice) would work.
- Git (for version control)
- Issue Tracker (for log changes and bugs)
- Maven (for dependency management and process automation)
- A metrics tool such as **Eclipse Metrics Plugin**, **MetricsReloaded**, **CodeMR**, (or any other alternative)
- PMD Maven plugin (for code quality check)

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*** (on an almost daily basis) use git features of ***branching*** and ***merging*** (not only on the last day before submission!).
    d) write your configuration files using ***YAML*** format. 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. 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.
5) **Readability**: make sure you write clean code, *exceptions* are correctly handled, and added comments to explain your code. Make your code "human-readable!".
6) **Quality**: check the quality of your code and outputs
    a) write unit tests using JUnit to test (at least!) the following functionalities: **open**, **save** and **search** (see details below).
    b) use code quality checking 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).

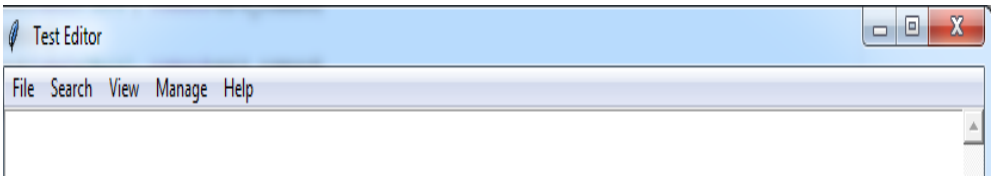
Make your own text editor!

1. The Text Editor

Your program is basically a standard text editor (or text processor) – something similar to Notepad, Atom (basic setup) 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 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 page of the editor.
    - **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 that you have used).
    - ***harder***: include a **PDF** conversation function to your editor so the file can be also saved into PDF format (for standard text files). Use an external library for this such as Apache PDFBox or OpenPDF.

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)
    }
}
```

- **Bonus Mark:**

  - Feel free to make your text editor intelligent! Why not try to process (and save) other file formats? The list of files that you can include is: rich text format (**RTF**) and OpenDocument Text (**ODT)** format.

  - OR: Use CI with appropriate testing from the start of the project.

Note : There is no specific requirement regarding which GUI library you should use. But try to make your program as cool as possible! There will be an extra mark for interesting ideas that have been implemented, but those should be reported and explained in the *Readme.md* file that should be submitted with the assignment.

2. Code Quality and Management

Once development is done, you need to 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

All submission is to be done using Stream:

- share your program on your ***private*** GitHub repository with us by sending a share invitation to the user **(user names will be provided soon)**

  This is to track commits on your git repository.

Include a Readme.md file in the top level of the project

The *Readme.md* is a text file with a Markdown syntax (this should be correctly formatted as a markdown syntax) 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 folders, what they contain.
3. for each student, a couple of the most significant git ***commit IDs*** that show the work of each individual member of the group.
4. any other interesting features you feel worth mentioning and that you think should be recognized in marking.

Who submits what?

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

- **member A**: submit (through Stream) a single compressed (e.g., zip or tar) file that contain the assignment (source file plus any executables)
  - **name the compressed file with both members' FirstName_LastName and ID numbers** (e.g. Xiaofeng_Liu-87878787-Susan_Jones-01002023.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.

Read more about **.md** files here
Markdown Quick guide
https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

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)	3
Correct implementation of the following functions: **New**, **Open**, **Exit**, **T&D** and **About**	2
Correct implementation of the **Select text, Copy, Paste and Cut (SCPC)** functions,	1
Correct implementation of the following functions: **Save**, **Search** and **Print**.	2
Advanced: **PDF** conversion function	1.5
Advanced: correct implementation of the following functions: **Open (read) .RTF and .ODT files**	1.5
Correct implementation of the following functions: **read source code files** such as .java, .py, .cpp or similar**.** Different syntax should be shown in **different colours.**	1
Appropriate use of git **FROM THE START OF DEVELOPMENT**	3
Appropriate use of issue tracking features to track changes/issues **FROM THE START OF DEVELOPMENT.**	2
Correct use of ***maven*** with all external dependencies correctly added	1.5
Correct use of configuration files (in YAML)	1.5
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) built-in CI with high quality unit tests	up to 2 marks
Total	**22 (max)**