Massey University

159.251 - Software Design and Construction

Assignment 1 - 2019

Deadlines and penalties

You must submit your final work using the stream submission system no later than **October 13rd 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 80% ...).

You are expected to manage your source code, this includes making frequent backups. “The Cat Ate My Source Code” is not a valid excuse for late submission.

Contribution to Final Grade **19%**

Read carefully as there are many parts that you should be aware of.

Overview

You are to work in **self-selected pairs** 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*** (feel free to select any issue tracker software that you would like to work with).

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 a free git hosting service for this assignment, and the ***repository must be private***. Bitbucket and Gitlab provide *free* private repositories. For Github, you can request an *academic licence* which will allow you to create private repositories.

IMPORTANT: Choose a conventional name for your repository (e.g. **251-Assignment1-FirstName1-FirstName2**). For example, if the first student is Sarah and second student is Lee, then the repository name should be (**251-Assignment1-Sarah-Lee**).

Please send invitations to join your repository to:

Shawn Rasheed: s.rasheed@massey.ac.nz

this is important and will be part of your assessment.

IDE independent: you may develop this in any IDE or code editor as your like! Tools included here are available for major IDEs, and also as Maven dependencies.

1. Tasks

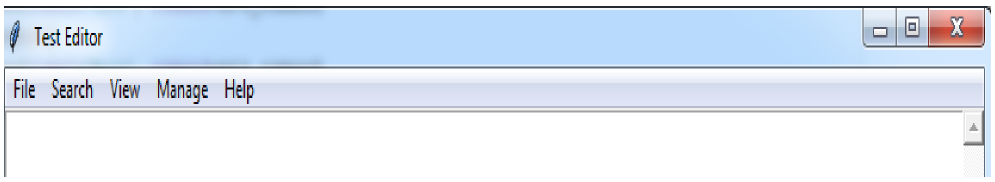
1) **Developing** a ***text editor*** program using Java – see details below.
2) **Source code and version control:**
   a) create and maintain a *git* repository on your local machine for your source code, and on a remote repository to provide a central server accessible to both members, and also accessible for marking.
   b) keep an audit trail of commits in the *git* repository. **These will form part of the marking.**
   c) Make sure that you ***actively*** use git features of ***branching*** and ***merging*** (not only on the last day before submission!).
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
   a) use ***maven*** to declare all dependencies, do not include any jar files with your submission.
   b) write your configuration files using ***JSON*** or ***YAML*** formats. You must submit at least one file in one of the two formats.
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 a 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 step 4).

2. Make your own text editor!

Your program is basically a standard text editor (or text processor) – something similar to Notepad, Atom (basic setups) 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

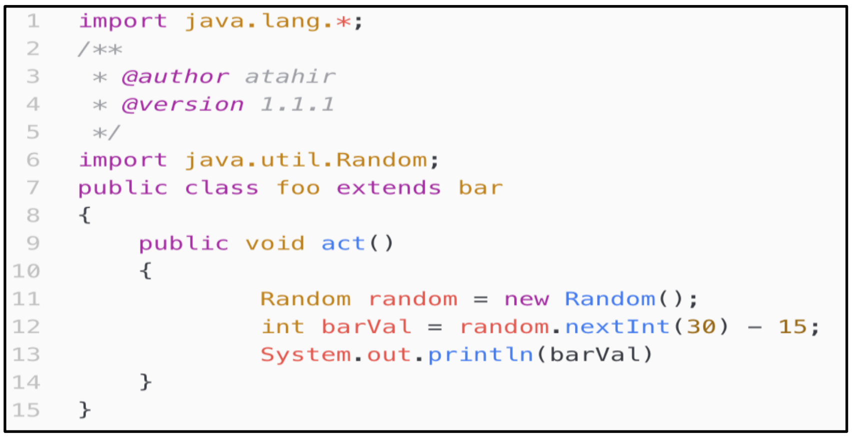


The menu should (at least) include the following sub-menus: File, Search, View, and Help

- 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 in 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).
    - include a **PDF** conversation function to your editor so the file can be also saved into PDF format (for standard text files).

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



```
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**) .doc, .docx and dot.
  - OR: Use CI with appropriate testing from the start of the project.

Once development is done, you need to report metrics data using a metrics tool. You may use the **Eclipse Metrics Plugin**, **or any other alternatives**. If you are using IntelliJ you may use **MetricsReloaded**, **CodeMR** or any similar tools. 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`

PMD maven plugin `https://maven.apache.org/plugins/maven-pmd-plugin`

Code Size (per class): Lines of Code (LOC) and Number of Methods (NOM)

Code Complexity: Cyclomatic Complexity and code coupling (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 included with your submission).

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.

Submitting your assignment

All submission is to be done using Stream:

- share your program on your ***private*** git repository (GitHub or Bitbucket) with us by sending a share invitation to the user **unshorn** (Shawn Rasheed).

  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 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 features you feel worth mentioning.

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
  - **name the compressed file with both members' FirstName_LastName and ID numbers** (e.g. James_Smith-87817172- 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, otherwise it won't let a mark be entered.

Read more about **.md** files here (http://stackoverflow.com/questions/5922882/what-file-uses-md-extension-and-how-should-i-edit-them)
Markdown Quick guide:
https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

Assessment

Your assessment will be based on the following criteria:

Criteria	**Worth**
Correct implementation of the **text editor** main window (which should also include a main menu)	2
Correct implementation of the following functions: **New**, **Open**, **Exit**, **T&D** and **About**	1
Correct implementation of the **Select text, Copy, Paste and Cut (SCPC)** functions,	1
Correct implementation of the following functions: **Save**, **Search** and **Print**.	2
PDF conversation function	1
Advanced: correct implementation of the following functions: **Open (read) .odt**	1
Advanced: 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.**	1.5
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***	1.5
Correct use of configuration files (YAML or JSON)	1
Check code quality and **report the specific size, complexity and other quality metrics.**	1
Overall code quality, including exception handling and comments to explain the code.	1
(extra) any ***additional*** adequate ***advanced features*** that are reported and explained e.g. built-in CI with high quality tests	1
Total	**20**