CSSE2002/7023 Assignment 3

Due: Friday, Oct. 27 15%

Goal: The goal of this assignment is to gain practical experience with implementing GUIs using the Model-View-Controller (MVC) design pattern and JavaFX.

Changelog:

- v1.0 Initial draft release.
- v1.1 Some changes to example GUI, fixing some typos.
- v1.2 Added character name field to example GUI, described UI components used, fixed typos in project code handout.
- v1.3 Added task to save the `CharacterDatabase`.

Problem description: In this assignment you will develop a GUI to manipulate the `CharacterDatabase`, using the methods implemented in Assignment 2, that is, `load()`, `save()`, `add()`, `remove()`, and `search()`. This extends the work you have done in Assignments 1 and 2. Notice that you won't be using the `Team` class in this assignment.

You will be given the skeleton for this assignment, with the Model-View-Controller classes and the FXML file for the GUI, to be edited using `Scene Builder`. You must not add any extra classes to complete this assignment. This will be found on Blackboard under Assessment.

Notice that changes have been made in the `Character` class in order to store the picture of the characters. The `Character` class only stores the name of the picture (that is, the path to the picture on disk), the picture will be loaded and displayed by the GUI, along with the character's other attributes, when a character is selected or searched.

In order to meet the requirements, the GUI must be capable of performing the following tasks:

- **Load** a collection of `Character` objects from the database, **(or create an empty database)** using the `CharacterDatabase`, and display the list/set of characters on the GUI window. The user should be able to provide the file name containing the database using the GUI. Characters must be displayed in a `ListView`, and must be sorted by *name* after loading.
    - **TIP1:** The easiest way to display a list of objects in a GUI is using the JavaFX `ListView` component.

  - **TIP2:** Use the `toString()` method to display the character objects in the `ListView`.
  - **TIP3**: You should take a look at the documentation of `ListView` in JavaFX, and also do some research about `ObservableList` in Java.

- **Save** the current collections of `Character` and `SuperCharacter` using the `ChracterDatabase save()` method.
  - Saving should only occur when the save button is pressed.

- **Add** new characters (or super characters) to the database using the GUI. The user must be able to input the `Character` or `SuperCharacter` attributes using JavaFX components, such as `TextField`, and add the new character to the database using an action button. The user must be able to choose whether a `Character` or a `SuperCharacter` is to be add to the database.
  - **TIP:** The character's picture can be add using the JavaFX `FileChooser` component. Notice that it is not mandatory to select a picture for the character, however, a default blank picture must be provided otherwise.

- **Remove** a character from the database using the GUI. The user must be able to remove a character either by providing the character's name to be removed, **or by** selecting the character in the `ListView`.
  - **TIP:** Even though the `ListView` component supports multiple row selections, for the sake of simplicity, the GUI must allow selection and removal of only one character from the `ListView` at a time (this is already the default configuration in the ListView control).

- **Search** a character by name, and display all information of the character along with its picture in the main GUI window. Information about characters must also be displayed automatically in the main GUI window every time a character is selected in the `ListView`.

- **Exceptions/Error Handling:** An appropriate error message should be clearly displayed (using an `Alert Dialog`) if any error occurs when using the GUI, for instance, if there is an error reading from the file, or there is any other error processing the database. The error message displayed to the user should clearly identify the reason for the error. It should include the detail message of the exception thrown to help the user track down the exact reason for the problem.
  - **TIP:** You should catch any exception thrown by the application in the Controller, and display the exception message from the `exception.getMessage()` method into a dialog box.

A directory of images (`src/images/`) has been provided for you to use in your testing, as well as an example database file (`src/database1.dat`). Note that the provided database file only contains two entries, so you may wish to do further testing by creating your own database files.

MVC Guidelines & General Notes

- You should not be storing any GUI elements (for example, anything from `javafx.*`) inside your `Model` class.
- All storing of variables and state (for example, your `CharacterDatabase` instance ought to be done in your `Model`
- You are only permitted to edit the `Model.java`, `Controller.java` and `view.fxml` files.

Submission

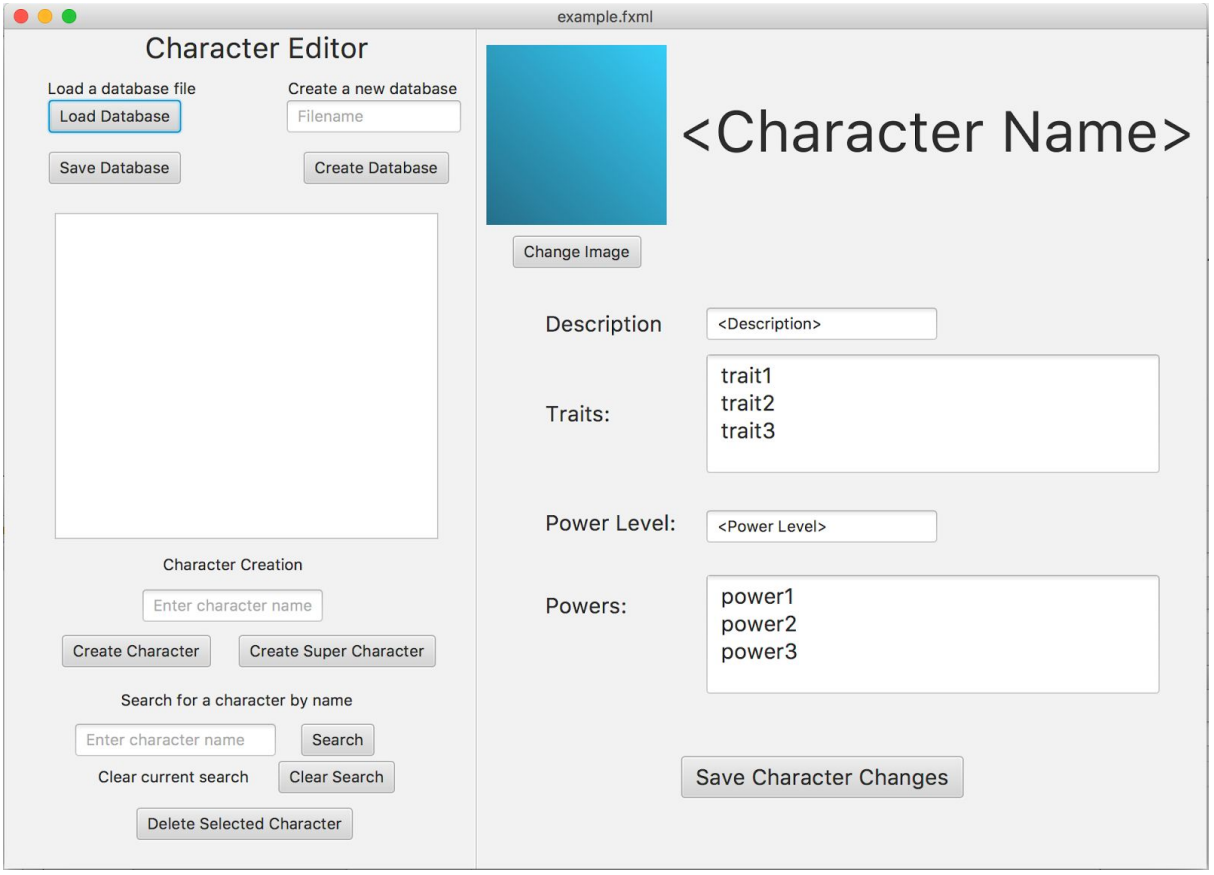
Assignment 3 is due on Friday 27th of October on Blackboard, and only the `Model.java`, `Controller.java` and `view.fxml` files are required to be submitted (others will be disregarded).

Marking

- Manual testing of the GUI: 50%
- Code quality: 40%
- Usability (user interface design): 10%

View Layout

Your main GUI window should look like the following:



User Interface Elements

The user interface elements used in the above are as follows:

- `Button` for all of the buttons
- `Label` for all of the textual labels (of all sizes - text size is larger)
- `TextField` for the 5 single line input fields
- `ImageView` for the view of the character (the blue square)
- `ListView` for the (currently empty) list of characters on the left

Note that when importing them in your `Controller` to import them from `javafx.scene.control`, **NOT** from `java.awt`.