CSE 216 – Homework V

Instructor: Dr. Ritwik Banerjee

This homework document consists of 3 pages. Carefully read the entire document before you start coding. This assignment is on event-driven GUI programming and unit testing. As such, you are being asked to do two things: (i) build a very simple reactive GUI using JavaFX, and (ii) build a test suite using JUnit.

Coding environment JDK 1.8 + JavaFX, JUnit 4.

Recommended IDE Intellij IDEA[1].

1 A simple GUI

In the first segment of this assignment, you must build a simple GUI that allows a user to enter three names along with their corresponding phone numbers. This interface must also include a button that says "Create Profiles". This button must initially be disabled (i.e., it should *not* be clickable before the user has typed any input text), and it must automatically become enabled and clickable once all the entries are non-empty. Once the button is clicked, the GUI window must make all the names and phone numbers un-editable.

The above description is subject to some stringent conditions on what names and phone numbers are to be considered "valid", and what behavior is required from the system if the user provides invalid input.

- **A name is valid** if, and only if, it consists of two words – one for the first name and one for the last name – and only the first letter of each word is in upper case. The entire name must also be no more than 20 characters in length.
- **A phone number is valid** if, and only if, it is a 10-digit number formatted as a string of the form "(###) ###-####", where each # is a single digit.

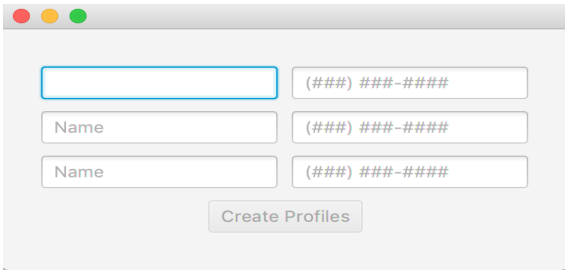

Figure 1: The initial GUI showing the text boxes, prompts, and the disabled and unclickable button.

1. The initial GUI must consist of (see Fig. 1)
   (a) three rows of two text-boxes each (the first for a name, and the second for the corresponding phone number), (5)
   (b) a button, centrally aligned, at the bottom (initially disabled, with the test "Create Profiles"), and (5)
   (c) initially empty text areas, with grayed out prompt text specifying what information is being requested in each box. Note that the prompt text must disappear if the text area is active. (5)

[1] You are free to use other IDEs, but the instructor and/or teaching assistants may not be able to provide much help if you get stuck with issues that are specific to your environment.

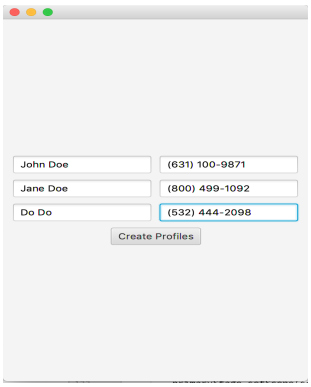

Figure 2: Window elongated

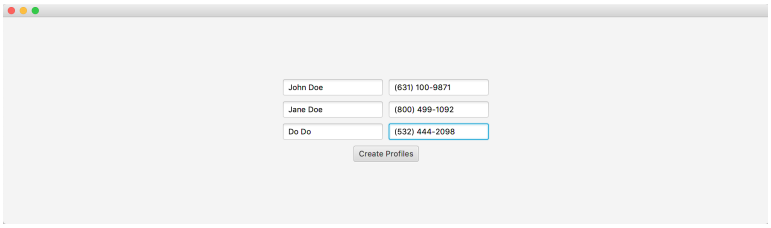

Figure 3: Window widened

2. The relative layout of the boxes and the button, as well as their central location within the window, must not get distorted if a user increases/decreases the window size. See Fig. 2 and Fig. 3. (5)

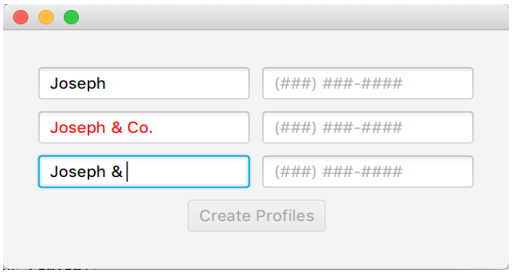

Figure 4: Checking the validity of information. The second and the third names are both invalid, but in the active text box, it is not yet rendered in red (yet).

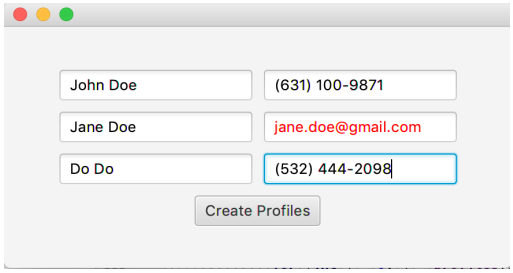

Figure 5: Button enabled when the user has provided information in all the text areas/boxes (even if some text is invalid, as shown in red).

3. The information entered by a user must be checked for validity, as defined earlier. This validity check must not require the user to do anything other than simply move beyond a particular text box. For example, if the user types an invalid name and just presses tab (or uses the mouse) to move to the next box. From a user's perspective, such a movement away from the box indicates that the information has been completely provided. In case the information is invalid, the text must be rendered in red immediately after (but not during) the information is entered. The validity checking (for names) is shown in Fig. 4. You must do this for names as well as phone numbers. (10)

4. Once all six text boxes have been filled with information, even if some of it is invalid, the button should become enabled and the user should be able to click it. This scenario is shown in Fig. 5. (5)

5. (a) If the user clicks the button while some information is invalid, a separate window must pop up and display the error message "INVALID INPUT: you have attempted to provide one or more invalid input(s). Please correct the information displayed in red and retry." The main GUI window never goes away, there is simply an additional pop-up window (it may partly or wholly cover the original GUI window) to display the error message. (5)

   (b) This pop-up window must have a button with the text "Close", and upon clicking it, the user must get the original window as s/he had before the pop-up came along. (5)

6. If the user clicks the button and all the information is valid, then

   (a) the GUI must immediately make all the text boxes uneditable, and (5)

(b) a separate window (just like in the invalid input case) must pop up with the message "The profiles have been saved and added to the database."[2] This window, too, must have a "Close" button. (5)

7. None of the figures show any title for your window, so add the title "Data Entry GUI" to your window. (5)

8. Just like the main GUI window, the error pop-up and the final notification windows must have a relevant titles too, e.g., "Invalid input error" and "Data Saved", respectively. (5)

   The main JavaFX application must be in a file called `DataEntryGUI.java`. You may use additional classes as well. But keep in mind that the grader will only run `DataEntryGUI.java`, so this must be the class extending JavaFX's `Application`, with the `public void start(Stage primaryStage)` method.

2 Unit Testing

In this component, you are required to use JUnit 4. Keep the following rules in mind:

- If a method being tested is in a file named `MyClass.java`, then the corresponding tests must be in a file called `MyClassTest.java`.
- Do NOT write tests for multiple methods inside a single test method.
- The method name (but not necessarily the whole signature) for the test method must be the same as the actual method.
- If you are testing for things that don't have a corresponding single method in the main code, it typically indicates lack of modularity for easy testing (for this assignment, though, you are not required to refactor your code to abide by such standards of code quality).

9. Write unit tests for the following:
   (a) The grayed-out prompt text for the name field(s) is/are, indeed, "Name". (2)
   (b) The grayed-out prompt text for the phone number field(s) is/are, indeed, "(###) ###-####". (2)
   (c) At least two test cases for invalid names and two test cases for valid names. That is, the tests must show that the validity checking for names is doing what it is supposed to do. (4)
   (d) At least two test cases for invalid names and two test cases for valid phone numbers. That is, the tests must show that the validity checking of phone numbers is doing what it is supposed to do. (4)
   (e) A test method to check that valid and invalid names are displayed in black and red, respectively. (4)
   (f) A test method to check that valid and invalid numbers are displayed in black and red, respectively. (4)
   (g) The "Create Profiles" button is, indeed, disabled until all the text areas have some text in them. (5)
   (h) If there is invalid information, the error box does indeed pop-up. (5)
   (i) If there is no invalid information, the text boxes all become uneditable, and the final box saying "Data Saved" does, indeed, pop-up. (5)

NOTES:

- As always, **late submissions** or **uncompilable code** will not be graded.
- Please remember to verify what you are submitting. Make sure you are, indeed, submitting what you think you are submitting!
- What to submit? A single `.zip` containing all your Java code. **Make sure there is NO package, etc.** in your submission. It should just be a `.zip` with your `.java` files in it. Also make sure that (i) your main JavaFX file is named `DataEntryGUI.java`, and (ii) you are using `JUnit 4`.

Submission Deadline: May 12, 2019, 11:59 pm

[2] For this assignment, you don't actually have to save any data; but saving such data would be a typical scenario for such a GUI-based data-entry system.