Homework 15 - Message Board

Authors: Shishir

Problem Description

Coming to the end of your CS 1331 career, you slowly start to reminisce about all you've done and how far you've come. Unfortunately, all good things must come to an end. As you look back on the semster, you decide that you want a place to share your memories with others who've gone through the same experience.

Because of this, you decide to create a message board for the class! Using your newly accquired JavaFX skills, you take it to yourself to make a chat room for everyone to share their thoughts on the class.

Solution Description

For this assignment, create the following file:

- `MessageBoard.java`

MessageBoard Class:

This will be the main point of entry into your JavaFX application. Your app should meet certain functional requirements:

There should be three main components to your application:

- A messages area where any posted messages can be seen.
- An area where the user can input their name and a message into text fields.
- A button that users can click to post their message to the messages area.

Specifications:

- The messages area should have a minimum height of 500.
- The windows should have a title that says "CS 1331 Message Board".
- The user should be able to tell which text field corresponds to the name and which text field corresponds to the message when typing in text. This can be done in many ways, including things like prompt text or labels for the text fields.
- Messages should be displayed sequentially in chronological order, with the most recent messages being closer to the bottom of the screen. The messages should be aligned vertically.
- A message should only be able to be posted if the name and message text fields are non-empty.
- Once a message is posted, the text fields should be cleared of any text.

Extra Credit:

For extra credit, display the messages in a ListView.

Sample Solution:

This is a sample solution that fits the problem description. This is not the only solution that fits the description, so don't be afraid to get creative!

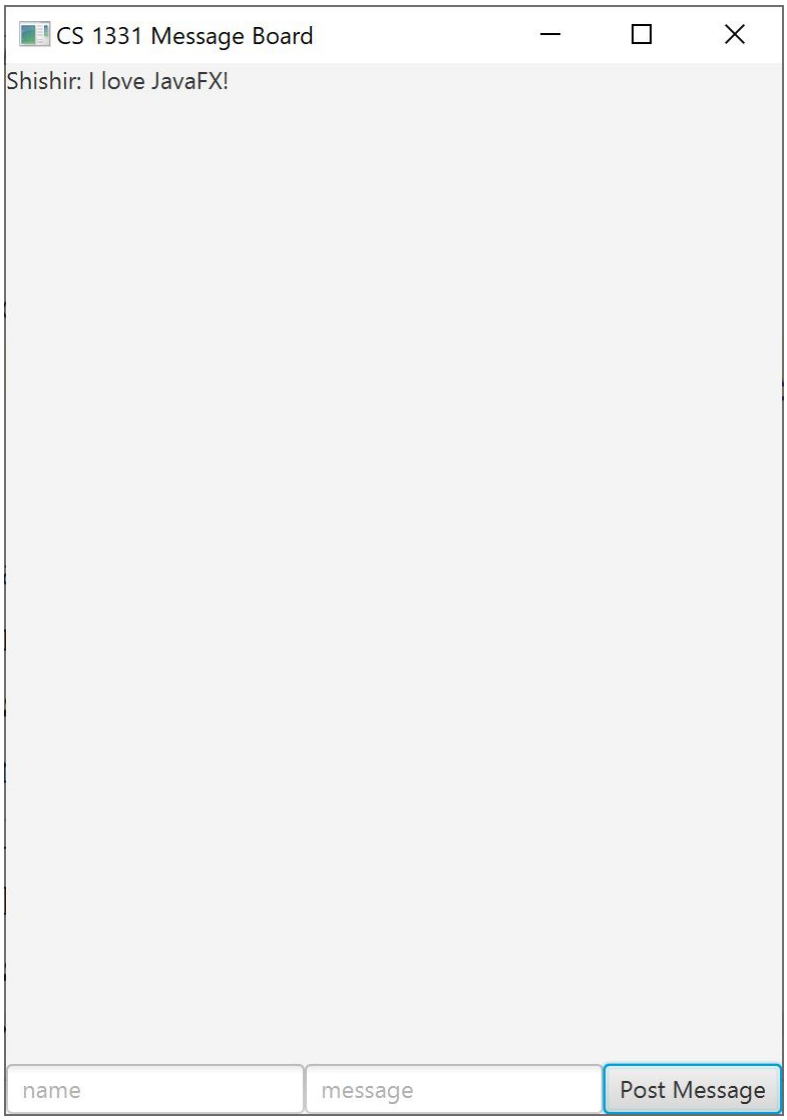

Figure 1: Before posting a message

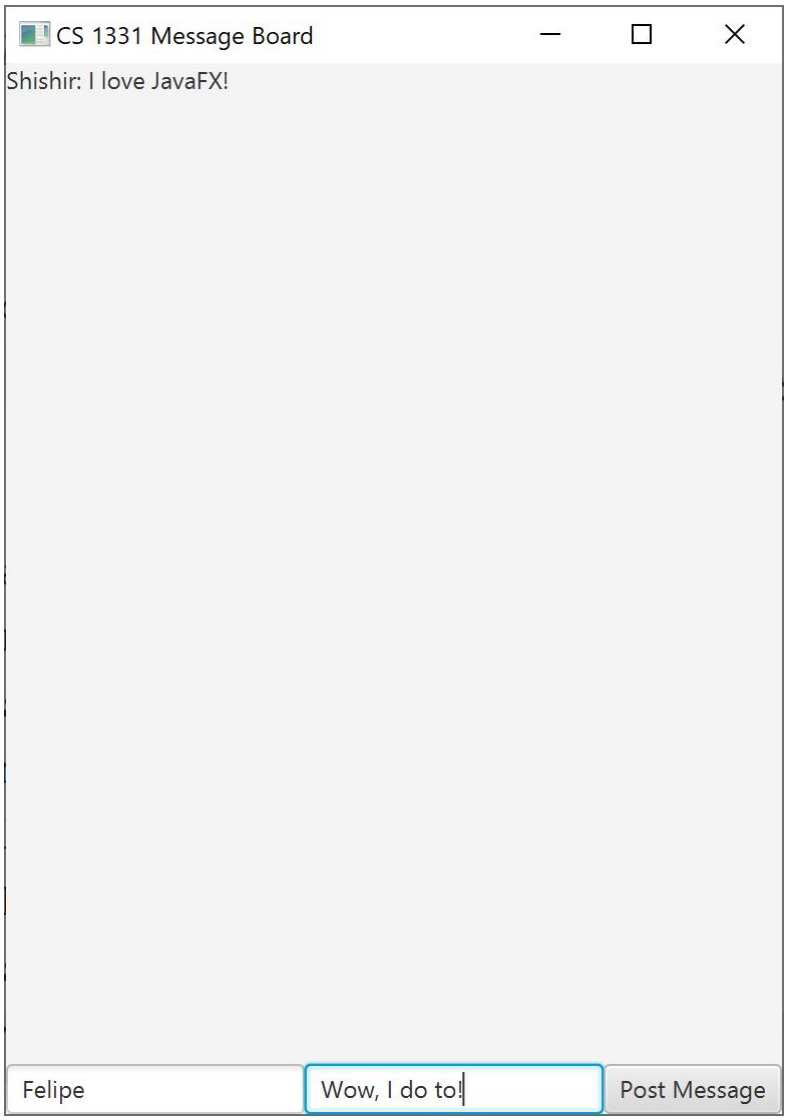

Figure 2: While typing a message

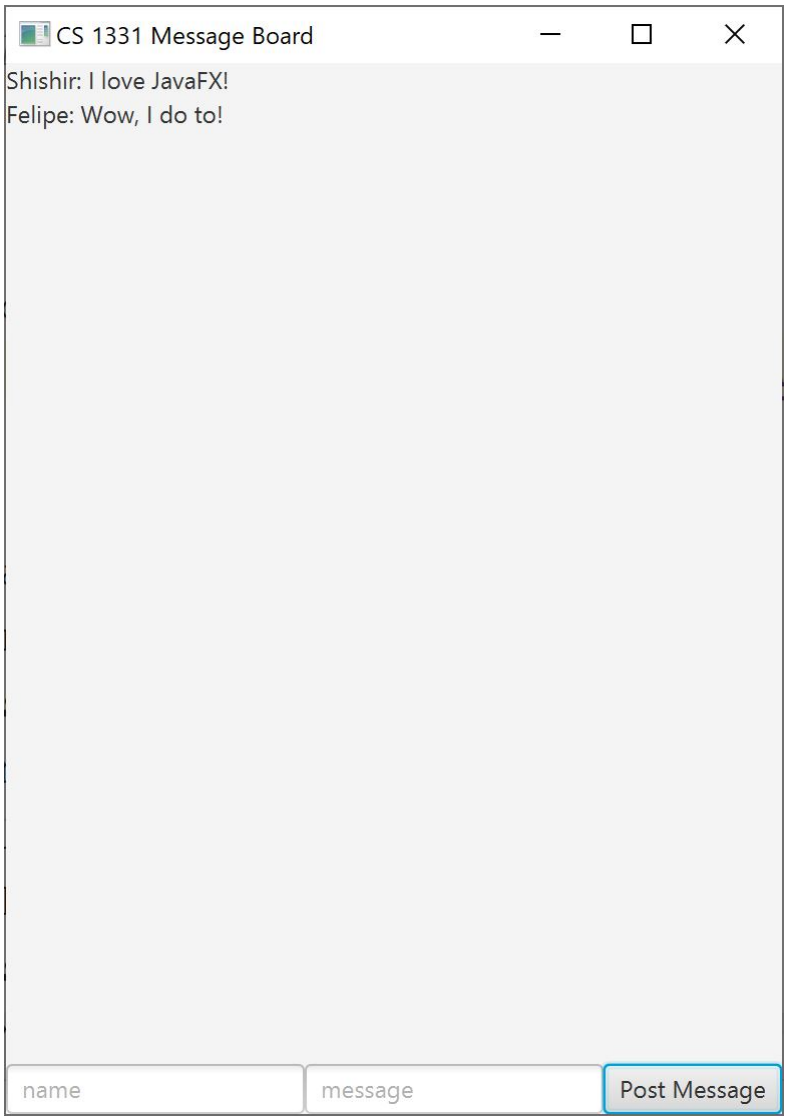

Figure 3: Posting a message

Hints and Tips

- `HBox` and `VBox` are your friend with layouts. Use them to liberally if needed.
- When posting messages, they are going to be vertically alligned. What layout helps us with this?

Rubric

- [10] It is clear where to input the name and message
- [20] User can input a name and a message
- [10] Message is only posted when there is information in the text fields and button is pressed
- [10] When message gets posted, text fields are cleared
- [10] Messages area has minimum height of 500
- [10] Correct title on window
- [30] All messages are displayed and are placed in chronological order
- [10] Messages are displayed in a `ListView`

Allowed Imports

You are allowed to import from any package that starts with `java` or `javafx`.

Remember that JavaFX is different that AWT or Swing. If you are trying to import something from `javax.swing` or `java.awt`, you are probably importing the wrong class.

Javadocs

For this assignment, you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and why they are awesome, the online overview for them is extremely detailed and helpful.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 * This class represents a Dog object.
* @author George P. Burdell
* @version 1.0
*/

public class Dog {

    /**
     * Creates an awesome dog (NOT a dawg!)
     */

    public Dog() {
    ...
```

```
    }

    /**
     * This method takes in two ints and returns their sum
     * @param a first number
     * @param b second number
     * @return sum of a and b
     */

    public int add(int a, int b) {
    ...
    }
}
```

A more thorough tutorial for Javadocs can be found here. Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@verion` tag included. The comments for a class should start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

Checkstyle can check for Javadocs using the -a flag, as described in the next section.

Checkstyle

For this assignment, we will be enforcing style guidelines with Checkstyle, a program we use to check Java style. Checkstyle can be downloaded here. To run Checkstyle, put the `jar` file in the same folder as your homework files and run

```
java -jar checkstyle-6.2.2.jar -a *.java
```

The Checkstyle cap for this assignment is **110 points**. This means that up to 110 points can be lost from Checkstyle errors.

Collaboration Statement

To ensure that you acknowledge collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment AT THE TOP of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Recall that comments are special lines in Java that begin with `//`.

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `MessageBoard.java`

Make sure you see the message stating "HW15 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

1. Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2. Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.