Distributed Collaboration - Assignment 1: Multi-View 1-User IM

Date Assigned: Wed Aug 20, 2014

1-View Target Date: Aug 27, 2014

Multi-View Submission Date: Sep 3, 2014(11:59 pm)

Objectives:

- Understand the use of observer pattern in user -interface toolkits
- Understand separation of UI and computation code
- Understand threads and synchronization (extra credit)

In this assignment you will implement the 1-User version of the IM/Editor project. The assignment is broken into two parts to allow you to incrementally build it. To make sure you do not procrastinate – there are two completion dates. The first one is a target date to help you pace yourself. The other one is the actual submission date.

The assignment involves the use of the Java Swing/AWT toolkit. Use the web to find tutorials on how to use ~~RMI~~ Swing and AWT – most books on introductory programming have sections on the toolkit. I have listed below some keywords to use in your searches. Of course, use Piazza for clarifications and other assignment discussion.

All assignments will most probably be changed after they are posted to clarify them or change constraints, and features. The changes will be highlighted using track changes,

Single View GUI

Implement a "single-user IM tool," along the lines shown in the first demo. You do not and should not slavishly follow the layout and user-interface shown in the demo. However, there are certain constraints.

1. The data or model displayed should contain the following (and possibly additional) components:
    a. an editable single-line topic text string. The string should be modelled as a list of characters rather than an atomic string. The reason is that we want to track the individual inserts and deletes into it, so that we can these incremental operations to other users rather than the whole string, which in turn allows

users to concurrently edit different portions of the string as you see in the operation transformation demos.

b. an editable single-line unaware-message text string.
c. an editable single-line aware-message text string (extra credit)
d. a non-editable multi-line string history ~~string~~ storing in each element a ~~separate lines~~message line ~~the messages entered using the message text strings~~a message entered by the user. The message liness~~s~~ should be stored in the order in which they are entered.
e. an uneditable status object (enum/String/int) showing whether the user has entered but not committed an aware message, is currently entering such a message, or has not entered an uncommitted message. The user is currently entering a message if the time that has elapsed since the last character insertion is less than some threshold time (I used 3 seconds). This status object perhaps makes little sense in the single-user case, as users know their status. It is intended to be transmitted to other users in the next assignment.

2. All of the data should be displayed in a single window or frame and the frame should be a Swing JFrame (and not an AWT Frame) object. The reason is that it is not possible/easy to draw a telepointer in an AWT classes such as Frame.
3. The unaware message should be displayed in an AWT TextField object (rather than a Swing JTextField) object. The reason is that it is not possible/easy to send JTextField text cursor positions correctly when you implement the shared window system, as you saw in the demo.
4. The topic and aware message should be displayed in a Swing JTextField object (rather than an AWT TextField) object. The reason is that it is not possible/easy to get information about character-level changes to a text field.
5. The status object can be displayed in any widget of your choice that is not editable by the user. In my demo I simply used a JTextField widget.

To implement this part of the assignment, you need to know about the widgets mentioned above and the associated listeners (ActionListener and DocumentListener). In addition, to implement the aware message, you need to know about threads, and the wait(time) and notify() methods. (Instead of wait() and notify()_ you could use sleep() and thread interrupt, which is not as elegant as interrupt throws an exception.)

One issue with using DocumentListener is that such a listener receives notifications both when the program changes the textfield and the user does so. Thus the listener does not know if it should change the model or not. One easy way to make this determination is to check if the textfield text is the same as the model text or of the same length – which would indicate that the change was made by the program and not the user. A related issue is that if you invoke the setText() method on the textfield, several insert/delete document events will be fired, during which the model text is not consistent with the textfield and yet you do not want to change the model text. To prevent this problem, you can set a boolean variable before setText() to indicate

you are changing the text field and unset it after setText(). Your listener should not set the model if the Boolean is set. These are not elegant solutions and show the need for a better API from Swing.

You can define a single Bean model (with properties of different types) for the various components described above. Ideally, the history and topic lists should use a common generic list class. The one provided in the ColabTeaching project should suffice as it is generic.

Multi-View UI

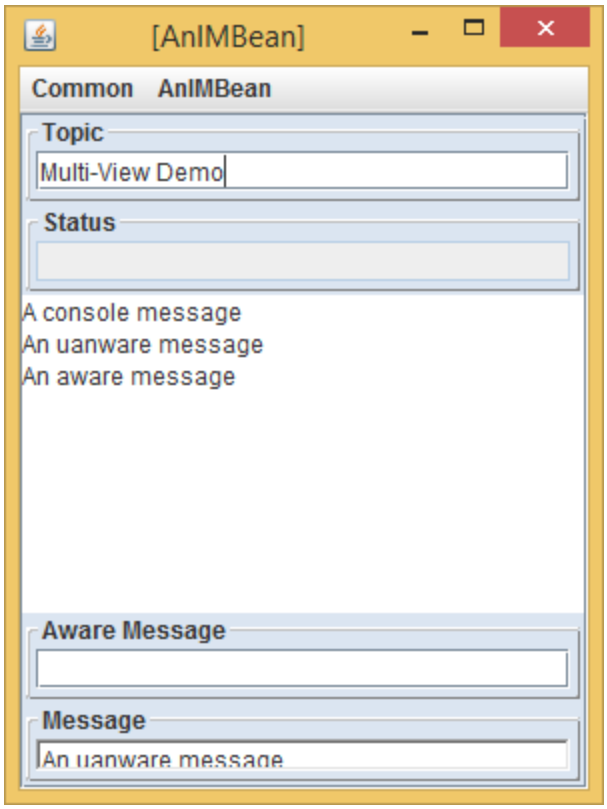
To ensure that you separate the user-interface and model code (important to make the application multi-user later) add a console view to the application. The console view should allow the user to:

1. enter a console line into the common message history displayed by the two UIs,
2. -view the history of messages by invoking the history command
3. -and quit the application by entering the quit command.

The console view is a slight extension of the one shown in demo 1.This UI is implemented standalone in the class examples – your job will be to stitch it to the GUI implemented in the previous art. Ideally, you should simply add the code shown in class to your project though you are free to change it. However, to aid grading, the console UI (prompts, commands, history output) should be exactly the same as shown below. The green lines are user input, the others are output. This will allow automatic grading of the assignment. You are free to add other commands to the user interface as the automatic grader will look for invariants among the output rather than the exact matches with an ideal output.

```
AliceEchoBean [Java Application] D:\Program Files\Java\jdk1.7.0_51\bin\java
Please enter an input line or quit or history
A console message
A console message
Please enter an input line or quit or history
An uanware message
An aware message
history
A console message, An uanware message, An aware message

Please enter an input line or quit or history
```



As we see above, the console view does not show the topic or the status messages, though you are free to extend it to display these data.

All of the data components mentioned in the first part will be implemented as Bean (readonly or editable) properties using getters and setters (for editable properties). The setters will fire property events observed by the GUI interactor. In addition, the history object will fire element-addition events observed by interactors for both UIs. If you use my history object, then element-addition events are fired for you.

To implement this part of the assignment you ~~will~~ may need to understand java.beans.PropertyChangeSupport, java.beans.PropertyChangeListener, and util.models. PropertyListenerRegistrar (optional and available from the util code on github and oeall22.jar)

~~Tracing~~

~~To provide better documentation and also support automatic grading, be sure to use the three Trace objects mentioned in class: ListEditInput, ListEditNotified, and ListEditObserved~~.

Extra Credit Demo

The extra credit requires you to support three states. Please each transition possible between each pair of states. Thus, show the transition from typing to has entered and has entered to typing. Similarly show the transition from no action to typing.

Submission

By the submission date, submit a link to a ~~YouTube~~ video (with an audio narration) showing your 1-user IM tool in action and also submit your code to Sakai. Please post the link as a private message in Piazza rather than an email (which is hard to keep track of) or a Sakai submission (the overhead of logging in to Sakai is high). Use the tag hw1videolink. If you are worried about privacy issues, free to use the other means, and post a Piazza message informing me you have done so.