Homework # 8

due Tuesday, January 20, 2:00 PM

In this assignment, you start with our solution to Homework # 7 and add a major new feature: each calendar is associated with a username (such as a pnather ID). One major task is recording at least ten unique refactorings.

1 Recording Your Refactorings

It is an important exercise to perform extensions on code only after a series of refactorings has made the extension easy to add (requiring, say, merely a new class). Such preparatory refactorings should “reform” the existing code, giving it new structure. They must not change the behavior and must not introduce “dead code”: it’s not a refactoring to add a new class or new method that is currently unused. In “real life” one will not use refactoring quite as heavily, but when first being introduced to a discipline, it’s best to use it exhaustively in order to learn how it works. You may be surprised at the benefits.

Do not try to “remember” what changes you did after the fact; you must log refactorings as you do them. If necessary, you can interpose non-refactoring changes between recorded refactorings, but such interruption (1) must be described and (2) probably indicates that you do not understand how refactoring works.

Please use a name (such as “Extract Method”) for each refactoring. Use the names from Eclipse, or from an “authority” (e.g., Martin Fowler); if you can’t find a name, make one up, and explain. Put your text in a plain (ASCII) test file refactorings.txt and place it in the root directory of your project. Please do not use RTF or MSWord.

As well as the name, please describe what the refactoring did, perhaps giving a snippet a code before and after to explain it.

After any refactoring that has major impact, especially if it had to be done by hand (Eclipse couldn’t do it automatically), you should perform a “run check” to make sure the program behavior has not changed. Record what refactoring steps are followed by run checks. Honesty is fine however; if a supposed refactoring turns out to have introduced a bug, go ahead and fix the bug and explain what needed to be fixed in the log.

You need to record at least ten different kinds of refactorings. If you’re not sure whether a refactoring is different enough from previous ones, then record extra refactorings. Our solution records a string of over twenty-two refactorings, including at least sixteen different kinds of refactorings. It doesn’t matter if a refactoring is very small (say renaming a variable) as long as it is recorded, (Of course, renaming a variable can only count once toward the quota of ten!)

We do not expect you record refactorings over the entire coding of the project. Start at some point early on and record until you are sure you have recorded at least ten distinct kinds of refactorings. Then you can stop this part.

Our solution to Homework #5 started as following:

1.1: Rename resetModel to updateView
KIND: RENAME METHOD
DESCRIPTION: resetModel is saying what we need to do when the calendar changes: reset the ListModel. But in general, this method is called when the view needs to be updated, however that needs to be done. So we use a more general name.
TESTED: no
(The first line gives a reference number and a one-line description.) This refactoring will not make sense for this assignment, of course.

2 Multiple Calendars

Currently (Homework #7) when the program starts, it starts with a view on an empty calendar. This should be changed so that a simple window with nothing but five buttons comes up (not even a title is needed!):

New  Ask for a username and then open a window an a new (empty) calendar for that username.

Open Ask for a username and then open a window on a pre-initialized calendar for that user username.

Connect Ask for a username and a hostname and connect the server on the host, requesting the given username’s calendar.

Serve Open up a dialog showing the current server statistics and a check box whether the server is currently active. Allow someone to check/uncheck the “server active” box, but don’t make any changes until they click “Apply” or “OK” (the latter also closing the window). There should of course also be a “Cancel” button to pull down the server.

Quit Close all windows and shut down the server and exit.

Only one calendar per user should be permitted.

The existing “File” menu should not have “Quit” anymore, but instead a “Close” menu item to just close the given window. The “New,” “Open” and “Connect” file menu items should operate on the given calendar as before (not putting up new windows).

The server should permit all open windows to be served when the server is active. The protocol will be changed by the client writing a UTF string object (writeUTF) indicating the username desired right after the initial bug flags int. Everything else should remain the same.

Remember to refactor first before implementing any new functionality!