

# Assignment 7: Save the Date – Planner System, part 3

## Due dates:

- **Implementation: Tuesday, Apr 09 at 8:59pm**
- **Self-evaluation: Wednesday, Apr 10 at 11:59pm**

## 1 Purpose

In this portion of the project, you will be building a controller for your planner and ultimately building fully working planner. Read through the entirety of this assignment before diving into coding, as the suggested designs here may influence how you structure your work.

## 2 Preparation: cleaning up prior code

You should likely have received design feedback about your model and/or view implementations. Incorporate that feedback, as well as complete any missing functionality from the prior assignments.

In the previous assignment, you implemented a lot of various actions that printed to the console or popped up a new frame. Nothing in particular needed to happen in response to those choices yet, but in this assignment we need to connect those choices to the controller. If you have not done so already, design a features interface describing those *user actions*, and enhance your view with the ability to add listeners for those actions. (Again, review [Lecture 15: GUI Basics](#) for more details on this approach.) This means as a first step, instead of having a listener print things or create a frame, have the class implementing your features callbacks do it instead.

## 3 Thinking about control-flow

As discussed in class, GUI controllers are *asynchronous*: they respond to events that may arrive at any time. Our planners are no different.

### 3.1 Listening and reacting to events

Making your controllers work will require thinking carefully about features interfaces. You have already considered what high-level events the view can publish, that the controller should listen for and respond to. These likely consisted of at least the following:

- using the XML file found by the client to add a user and their schedule to the system
- saving all schedules in the system to the folder chosen by the client to their own XML files
- creating a new event
- modifying an existing event

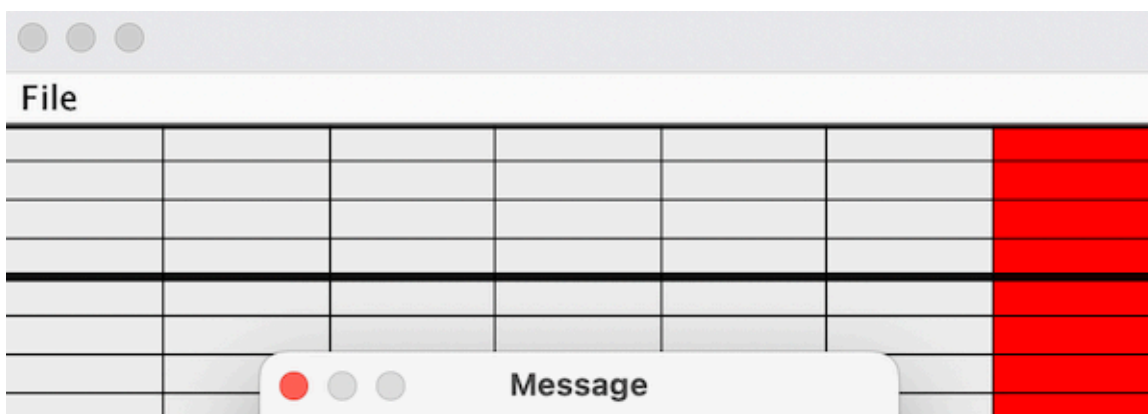
- removing an event
- creating an event frame for creating/modifying an event
- switching the currently viewed user

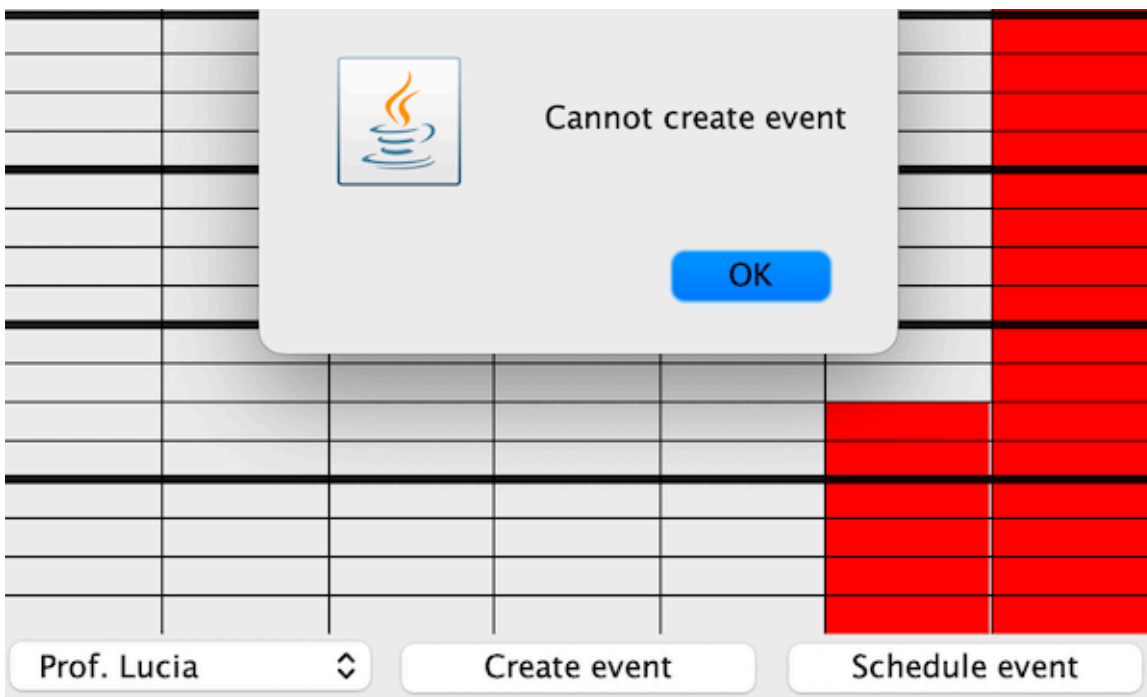
We needed these high-level events because they could happen at any time, so the controller had to wait for them to occur before it could respond.

Your controller should now be listening for calls to those methods from the view as part of the command callback design pattern.

## 4 Designing the controller

- The controller implementation will need to take in your view for the entire planner.
- The controller will need to register itself as a listener of view actions somehow (i.e. the callback for all of your features).
- The controller should ensure that its view stays up to date after any model modifications it makes. Reread the discussion specifically about `toggleColor` in [Lecture 15: GUI Basics](#), as the design concerns there might be similarly applicable here.
- Your model is likely capable of throwing exceptions when events cannot be added or if loading a schedule into the model fails. Your controller and/or your view must somehow handle these exceptions and make them visible to the user; it is not acceptable for them to silently fail and leave a warning in the console. For example, you can use `the showMessageDialog method` to display the error message to the user:





Make sure to test your controller thoroughly.

**Hint:** To test the controller, you will need a view that calls to your callbacks. However, we cannot use our GUI because JUnit cannot interact with it. What if you made a fake version of the view that is NOT graphical for the purposes of testing? Is that the only fake version of a component you will need?

## 5 Running the planner

### 5.1 Combining the controller and view

In the previous assignment, we built a placeholder `main()` method that simply created a view for the model. Now, we need to create a controller as well to make the entire visual program responsive.

```
package cs3500.planner;

public final class PlannerRunner {
    public static void main(String[] args) {
        YourModel model = ...create an example model...
        YourView view = new YourView(model);
        YourController controller = new YourController(view);
        controller.launch(model);
    }
}
```

## 6 Scheduling events

We are missing one piece of functionality after all of this work: automatically scheduling events. We will allow the user to enter the duration of the event instead of giving a specific

start and end time for the event. Our program will then decide when to schedule the event based on some criteria. You will be implementing these scheduling strategies into your program and allowing a user to choose them as a command line argument.

## 6.1 The scheduling strategies

We will build in two strategies for scheduling events. These strategies will find the start and end times (day and time, to be specific) of the event. The two kinds of strategies you *must* implement are as follows.

- "Any time": This scheduling strategy will find the first possible time (starting Sunday at 00:00) that allows all invitees and the host to be present and return an event with that block of time.
- "Work hours": This scheduling strategy will find the first possible time from Monday to Friday (inclusive) between the hours of 0900 and 1700 (inclusive) where all invitees and the host can attend the even and return an event with that block of time.

Make sure to test the strategies thoroughly.

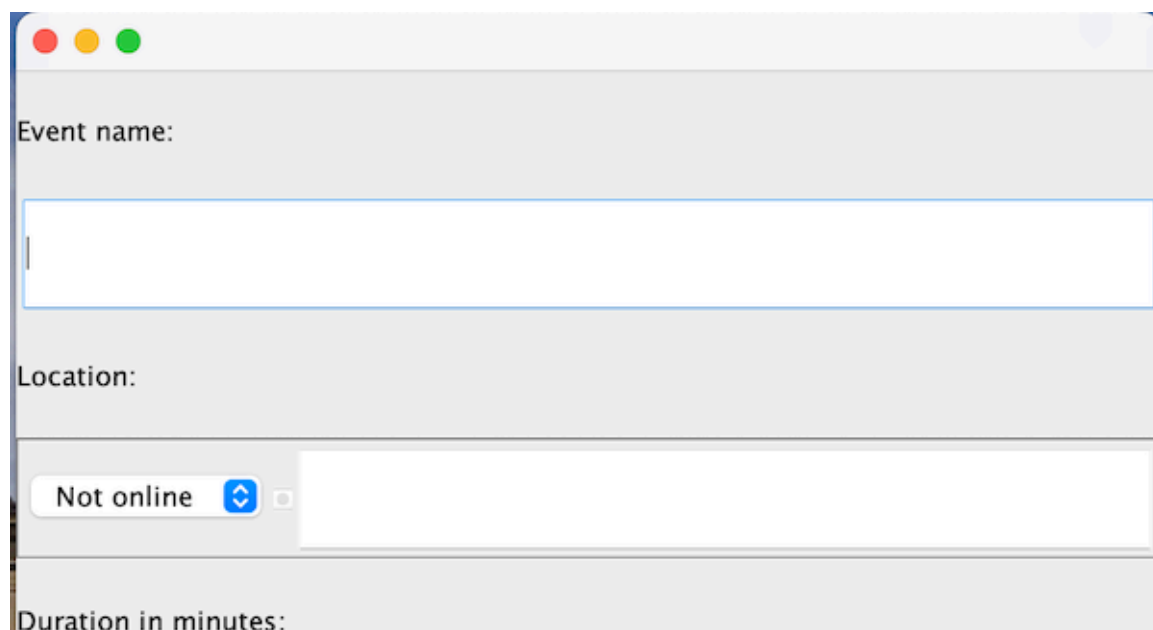
**Hint:** Try a brute force approach. That is, don't try to pre-optimize your solution and instead keep your solutions simple.

## 6.2 The scheduling frame

Make a new frame that is *almost* the same as the event frame from Hw6. The differences visually are as follows



- Instead of places to take in the starting day/time and ending day/time, you will have one place to take in the number of minutes representing the duration of the event.
- Instead of having multiple buttons to create/modify/remove the event, you will have one to simply schedule it.

Here is an example of what that can look like



Event name:

Location:

Not online  

Duration in minutes:

90

Available users

Prof. Lucia  
Chat  
Student Anon

Schedule event

### 6.3 The expected behavior of the new frame

The main system should have a button exposed to schedule an event. Clicking on that button should open up the scheduling frame with no values entered. Once the client enters the relevant data and clicks button on the scheduling frame, the program should attempt to find a time to schedule that event. If it fails to schedule, it should let the user know somehow.

### 6.4 Configuring the strategy

As with SameGame, we would like to be able to configure our planner. However, the only customization is the strategy for scheduling events.

Your command-line here does not have to be particularly elaborate: at minimum, it should expect one string argument describing the scheduling strategy to use, for example, `"anytime"` or `"workhours"`. Document whatever command-line arguments you choose. Augment the scaffold code above for `main` to take into account the command-line and cleanly configure the program accordingly.

Review [instructions from SameGame](#) on configuring IntelliJ to run your program with command-line arguments

## 7 What to do

1. Create the interface for high-level view events as discussed above. Make sure you have clearly documented the purposes of that interface and its methods, to guide you in the next implementation stages. Implement the ability for your view to add an object that listens to for those high-level events.
2. Design a controller that takes in a view. It should register itself as a listener for your features interface, and should mediate between the view and the model as needed.
3. Create the interface for your scheduling strategies as discussed above. Make sure you have clearly document the purpose of that interface and it's method(s). Then implement the required strategies

implement the required strategies.

4. Update your view with the new scheduling frame.

5. Update your controller so it takes in the desired strategy as part of the method that starts the program.

6. Update your high-level view events to account for the new scheduling feature. Have your controller use the strategy to attempt to schedule the event.

7. Update your `main` method as described above to use command line arguments.

8. Update your README file to include explanations of all the new classes you've designed. Be sure to include a "Changes for part 3" section documenting what you changed from your initial design or from the previous assignment.

## 8 What to submit

- Submit all your source and test files so far
- Submit your updated README file
- Submit the three screenshots demonstrating your view works as intended. **These may be the same as last assignment and that is okay.**
- Submit a **new** screenshot with your scheduling frame to demonstrate it works as intended.
- Submit a JAR file (with extension `.jar`) that can run your program. (See the instructions in **Assignment 6** for how to create a JAR.)

## 9 Grading standards

For this assignment, you will be graded on

- the design of your view and strategy interfaces, in terms of clarity, flexibility, and how plausibly they will support needed functionality;
- how well you justify any changes made to your model;
- the forward thinking in your design, in terms of its flexibility, use of abstraction, etc.;
- the correctness and style of your implementation, and;
- the comprehensiveness and correctness of your test coverage.

## 10 Submission

Please submit your homework to <https://handins.ccs.neu.edu/> by the above deadline. Then be sure to complete your self evaluation by its due date.

## 11 Extra credit opportunities

Here are some opportunities to get some extra credit on this assignment. These can only be used with this assignment, meaning they are due at the same time as your assignment. Therefore it is possible to get over 100% on this assignment! However, to get any of the extra credit, the new feature must work *perfectly*.

If you implement any of the following, add a section to your README called "Extra credit" and explain what extra credit you did.

### 11.1 Resizable views

Making views resizable is a nice convenience feature. For event frames, Swing allows them to be resized pretty easily thanks to the `LayoutManager`. However, the main system frame with its panel for viewing schedules is not so lucky. As a result, your main system frame did not have to be resizable for Hw6.

For extra credit, make the main system frame completely resizable and adapt to that new size. Resizing this frame with schedules already visible should show the same events correctly. You can see an example of this with the error message picture: that frame was resized to be smaller, resulting in the more compact look.

Explain the README under the "Extra credit" section exactly what changes you made to your code to allow for resizability.

### 11.2 A lenient strategy

In this entire project, we always make sure everyone invited to an event can attend the event. In other words, there can be no conflicts for any invitee. However, as programmers we can make our system more lenient when it comes to automatically scheduling these events.

To that end, create a new strategy that creates an event in the first time block where the host and at least one other invitee can add that event to their schedule. If there is more than one non-host invitee that can add the event to their schedule, then they must also be invited. The strategy must find a block between Monday and Friday (inclusive) from 0900 to 1700 (inclusive).

Explain in the README where to find the code for this new strategy, what the command line argument for that strategy is, and where to find all tests related to that new strategy.

### 11.3 How these are graded

Functionality for each extra credit item is all-or-nothing. If it does not work perfectly, you will gain 0 extra credit points for that extra credit item. If it *does* work perfectly, you will gain some points and then you will be further graded on the design of the extra credit.

We will determine how many points of credit this is worth after you submit the assignment.

