CS193E: Assignment 5
PersonalTimeline II

Due Date

This assignment is due by **11:59 PM, February 13.**

Assignment

This week you will only do very basic drawing - next week we'll work more on the layout of our timeline. For now we're interested in getting the basic flow of data between model, view, and controller.

There are a number of parts to this assignment. Here's a rough outline of what you'll need to do to complete the assignment successfully:

1. Create an NSView subclass, TimelineView.
2. In your document nib file, increase the window size and add a custom view. Make the custom view a TimelineView.
3. Create a TimelineItem class. This class is a subclass of NSObject, it is not a view, but will encapsulate the geometry and drawing code for each timeline item.

   The TImelineItem class should hold the bounds of the item and the title of the item (Later we will extend it).
4. Implement basic drawing. The timeline item should draw at a minimum a box and a title within the box. Draw each TimelineItem from top to bottom.
5. Make sure TimelineView updates when changes are made to the table view - additions, removals, and edits to the value of an event. This also includes updating for undo/redo operations.
6. TimelineView Selection - implement basic hit testing. If there is a mouse down on a timeline item, it is selected. Its background color should change to indicate it is selected. The TimelineView should vend a 'selectedIndex' method and a setSelectedIndex: method. When a click occurs in the timeline view outside of any timeline item, all items should be deselected. The notification should still be sent, and following NSTableView's example -selectedIndex should return -1.
7. The TimelineView should define a TimelineViewSelectionDidChange notification and post it at appropriate times.
8. Keeping selections in sync. Register for notifications on both the timeline view and table view to find out when selections have changed. Keep the selections of the two views in sync.

Testing

Make sure the following work in your application (these are the things we will test to determine your grade).

1. Your project should build without errors or warnings.
2. Your project should run without crashing.
3. The timeline view should display all timeline events from top to bottom. Each timeline event should be represented by a rectangle with the title of the timeline event drawn within it.

4. When you add or remove a timeline event, the timeline view should update appropriately.
5. When you finish editing a timeline event in the table view, the timeline view should update as well.
6. Undo and redo operations of all types should also be properly reflected in both the table view and the timeline view.
7. Clicking a timeline item in the timeline view should select it. This should also select the appropriate item in the table view.
8. Selecting an item in the table view should also select the appropriate item in the timeline view.
9. Clicking an area in the timeline view without a timeline item should deselect all timeline items. The table view's selection should stay in sync.
10. All previous features of the application should continue to work.

Hints

You may find it easier to handle drawing items from top to bottom if you use the flipped coordinate system in the TimelineView. To do so, implement -isFlipped to return YES.

First focus on displaying TimelineItems before moving on to hit testing, selection, and finally keeping selections in sync.

There are a number of ways of getting information into and out of the view. One way is to take an approach similar to a table view. In this approach, a timeline view would have a data source and define an informal protocol to be implemented by the data source. Methods in the informal protocol might include asking the data source for the number of items, and for asking for various values for each item.

The table view has the -reloadData method. A similar method on TimelineView would be very helpful.

Troubleshooting

Some common problems when using a table view are forgetting to hook up the data source (either in Interface Builder, or programatically) and misspelling the name of a data source method. If you are using Key-Value Coding, be certain that the table view identifier string set in Interface Builder is the correct key for the property that column is displaying.

There are three common problems developers run into when working with notifications. The first is simply *forgetting to post the notification*. You might have all the code set up correctly for being an observer but if nobody ever posts the notification, your code won't get called. This includes making sure you post the *right* notification! An easy mistake is to have a typo in the name of the notification. To avoid this pitfall, instead of hard-coding an `@"..."` string in the places where you post and register observers, put the string into a global and use the global instead. That ensures that everybody uses the same string value for the notification. For an example of this, look at the bottom of `NSWindow.h` where there are global strings for a bunch of notifications. Of course, you would simply use `extern` instead of `APPKIT_EXTERN` when declaring your constants.

The second common problem is the converse of the first: *forgetting to register as an observer* of a notification. Nobody cares if there's a tree falling in the woods if you haven't registered for a

notification! Also, make sure the observer actually implements the method that you specified as the "`selector:`" when adding the observer.

The third, and by far the worst, problem is *failing to remove observers before getting deallocated*. Remember that `NSNotificationCenter` doesn't retain observers. If an observer is deallocated but is not removed from the notification center, then the next time a relevant notification is posted, the notification center will send a message to a deallocated object which will usually crash your program. In your `dealloc` method, make sure to remove objects from the notification center.

Extra Credit

Keep in mind that extra credit only applies if the required behavior outlined above is working. Don't spend time on extra credit until you've got the basics down.

- Æsthetics matter: a particularly attractive, innovative, or clever design of the interface is always appreciated. (If you want consideration for extra credit in this case, you must let us know when you submit your project.)
- Having a -reloadData method that tells a view (like TimelineVIew) to reload all of its data, and therefore likely redraw itself, is an effective but heavyweight way of getting a view to update. Add a method to TimelineView that tells it to reload data for a specific item. This should pull the current information for that item, and redisplay only that item in the timeline.
- Use rounded rects. Rectangles are lovely in their own right, but very often in user interfaces we encounter rounded rects. Create a category on NSBezierPath that allows a rounded rect to be created (perhaps specifying the bounds as well as the radius of the corners). Use the rounded rect to draw each timeline item, and also to perform hit testing to see if an item is selected.
- Any of the extra credit from Personal Timeline I is fair game for this assignment (assuming you didn't already submit it previously — no double dipping!).