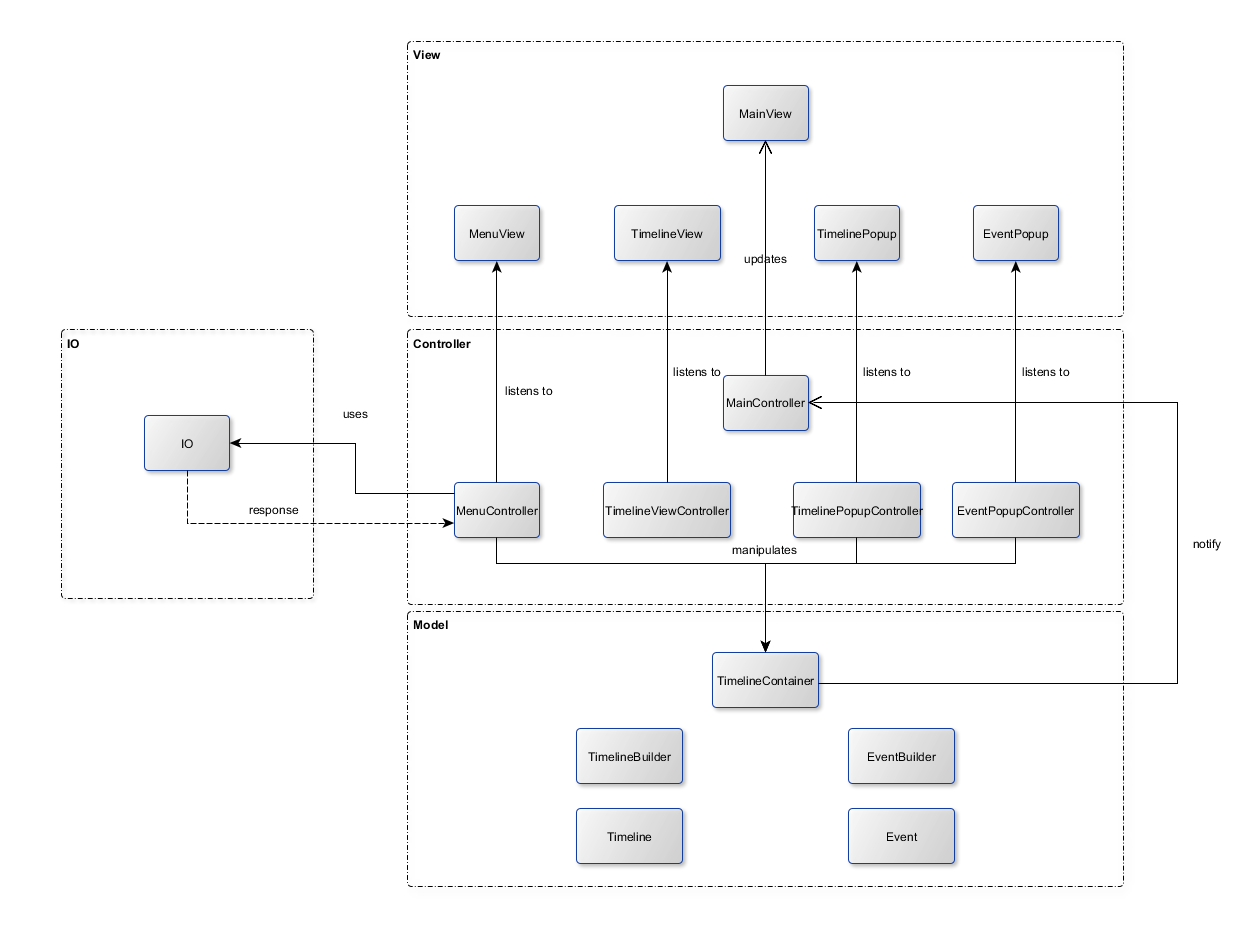
## Classes



### View Classes

* **MainView** – The root pane of the application. Contains all other views.
* **MenuView** – The menu at the top of the window, has a drop-down menu for selecting timeline and buttons for creating a new timeline, saving and loading a timeline and deleting a timeline.
* **TimelineView** – The part of the main window that displays the timeline and its events. Has a button for adding an event to the current timeline. Is also the were information about an event is displayed when the user hovers over an event with the mouse.
* **TimelinePopup** – The popup where the user inputs information about a timeline he/she wants to add.
* **EventPopup** – The popup where the user inputs information about an event he/she wants to add or edit.

### Controller Classes

* **MainController** – Manages the other controllers.
* **MenuController** – Handles logic for user interactions with the menu bar.
* **TimelineViewController** – Handles logic for user interactions with the timeline view.
* **TimelinePopupController** – Handles the logic for the input fields and OK/Cancel buttons in the add timeline popup.
* **EventPopupController** – Handles the logic for the input fields and OK/Cancel buttons in the add/edit event popup.

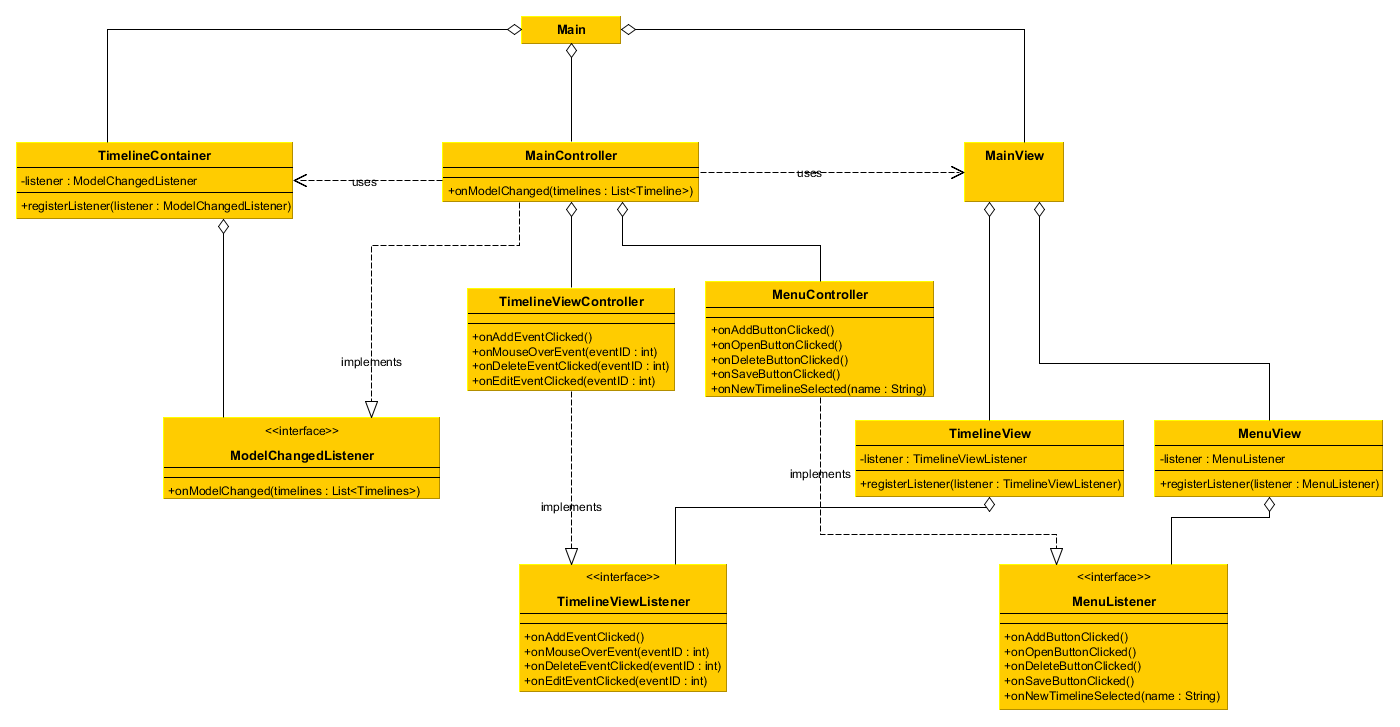
### Model Classes

* **TimelineContainer** – Stores a list of the timelines that are currently opened.
* **TimelineBuilder** – Creates timelines from the timeline information inputted by the user.
* **EventBuilder** – Creates Events from the event information inputted by the user.
* **Timeline** – Object that represents a timeline. Has fields for the timeline properties and a list of events that have been added to the timeline.
* **Event** – Object that represents an event on a timeline. Has fields for the event properties.

### IO Classes

Not yet decided.

## Class Relations



Since we wanted the view and model to not be directly related to any of the classes in the other components, we used a variant of the *Observer Pattern* to communicate user interactions from the model to the view and notify the view of changes in the model. In our design, the controller classes TimelineController and MenuController implements the interfaces TimelineViewListener and MenuListener and then gets registered as listeners in TimelineView and MenuView. Since TimelineView and MenuView does not know anything about the controller classes except that they implement the methods defined in the interfaces, this allows the controllers to be notified of user interactions without exposing any other details about its implementation to the view classes. The model notifying the controller of changes works the same way: MainController implements an interface, ModelChangedListener, and gets registered as a listener to the TimelineContainer. Whenever a change occurs with the timelines in the TimelineContainer, it notifies the listener via the onModelChanged method.