Kurt Andres, Daniel Conroy, Leanne Miller & Andrew Thompson

Software Development

Phase II Writeup

**Guide Through the Code:**

The main method to run the application may be found in model\Driver. By running this main method, the TimelineMaker application starts by initializing the storage, graphics, and gui objects. It then loads the existing timelines if any exist in the current directory and displaying the first selected category. This is done through the Storage package, this utilizing XML to save the timelines and their constituent parts.

Once the program is started, timelines can be created, edited, or removed from the application. Categories may be added to these timelines, which similarly may be edited or removed. The Category class possesses a Default Category that is impossible to delete such that it is the default location for events, and it is to this category that all events are sent if their own category is removed. Events are similarly added to Timelines, each having reference to a Category and giving a reference of themselves to the Category. Events are divided into Atomic Events and Duration Events, sharing much of the code in a parent class Event.

The GUI is written entirely in Swing and is centered on the MainWindow class which is a JFrame of the main edit window. The MainWindow class stores the many buttons and panels seen in the window, including the graphical DisplayPane. The DisplayPane houses the JavaFX JFXPanel used for graphical rendering. In addition, TimelineMaker’s action listeners handle user input either by invoking appropriate methods in TimelineMaker or by creating an appropriate PropertiesWindow object, which then call a TimelineMaker method when they are closed. Since the TimelineMaker mediates the interaction between all of the parts of the application and thus models the state of the application, the MainWindow is not responsible for storing any timelines or selected-timelines or anything else like that. Instead, it stores a reference to the TimelineMaker object and acts on it.

The graphics are controlled by timeline renderer which renders three main elements: the timeline decorators(axis, title, and time-period labels), the atomic events, and duration events. This is done through JavaFX and css styling. The EventAPI is used to get the info of each event which is then translated into a JavaFX label which is then added to the group before rendering to the timeline. Atomic events are rendered first, then the timeline, then duration events. This is because the entire rendering process takes place from top down so as to know current locations for rendering the events and to separate atomic vs duration events by the timeline axis. The code keeps track of a variable called pushdown, which is merely the current y offset from the top. This allows for the rendering of more events without stacking them on top of eacho ther. The events each have an action listener for selection which than allows for the editing of that particular event to be invoked. The events also have a hover over listener that changes the event category color indicator slightly darker when the mouse passes over, this is controlled through the isHovered method of each label. Also note, the actual timeline is drawn merely as a line from coordinates under a JavaFX label so as be easier to manipulate.

**Users:**

In general, users were most concerned with the display of timelines and the graphic user interface. Items requested were such things as a wiki-esque sort of link system between timelines, better artistic rendering of events, and access to links to the outside world, such as a link from an event about the reformation to a Wikipedia page about the reformation. In addition, many users suggested cleaner, more generous (in terms of error-handling and ease of use) GUI and more attractive rendering of the timeline.

**Decisions:**

Because the user’s requests were mostly made concerning the graphics, and seeing as this was not the primary goal of the project, we thought that these issues were most applicable for the third phase rather than the second. Therefore, in this phase we focused on cohesive and encapsulated code, maximizing reuse, and providing the basic for the storage of categories, colors, and preparing the ground for basic rendering. Time was spent attempting to store data online, but for time purposes we decided to store in the local repository.

Users also advocated for simplicity of look, and ease of reading when it comes to timelines and a minimalist look as opposed to an overly complex and hard to read approach. Users favored Kurt and Conner’s original rendering of timelines so we stuck to a basic feel.

**Individual Responsibilities:**

Kurt Andres – I implemented the timelineRender class and the duration/atomic event label classes to handle the rendering of time, events, and decorators. I re-worked the coordinate system, added axis lines, labels, selectors, and hover effects. Much time was spent researching dynamic css color picking to allow category selection and de-selection colors to try and group categories by color while still maintaining the ease of use associated with JavaFx labels. The goal was to use labels so everything could be selected and (in the long run even moved) by the mouse. In addition a hover over tooltip was added to provide the info about each event. This was implemented in each label.

Daniel Conroy – I implemented the Category class and its interaction with the rest of the application. I also modified timelines and events to accommodate categories, while also refactoring timelines and events to a degree. I made refactoring changes to event labels and aided to a degree in the saving and loading of timelines. I participated in discussion over the inclusion and structure of categories as they relate to the rest of the data structures.

Leanne Miller

Andrew Thompson – I was responsible for updating the GUI for the new Category data-structure. I wrote the entirety of the new CategoryPropertiesWindow class and also had to modify the rest of the GUI classes to accommodate the addition. Furthermore, I wrote all of the methods in TimelineMaker for handling category selection, addition, deletion, and editing. I spent a lot of time trying to implement a JList of CheckBox items for selecting multiple categories, but all of my efforts in that regard were frustrated, so I decided to instead implement a single-selection JComboBox and a toggle button to allow users to choose to display all the categories or only the selected one in the graphics window. In order to accomplish this, I also had to write the CategoryRender class, which renders just a category of elements rather than the entire timeline.