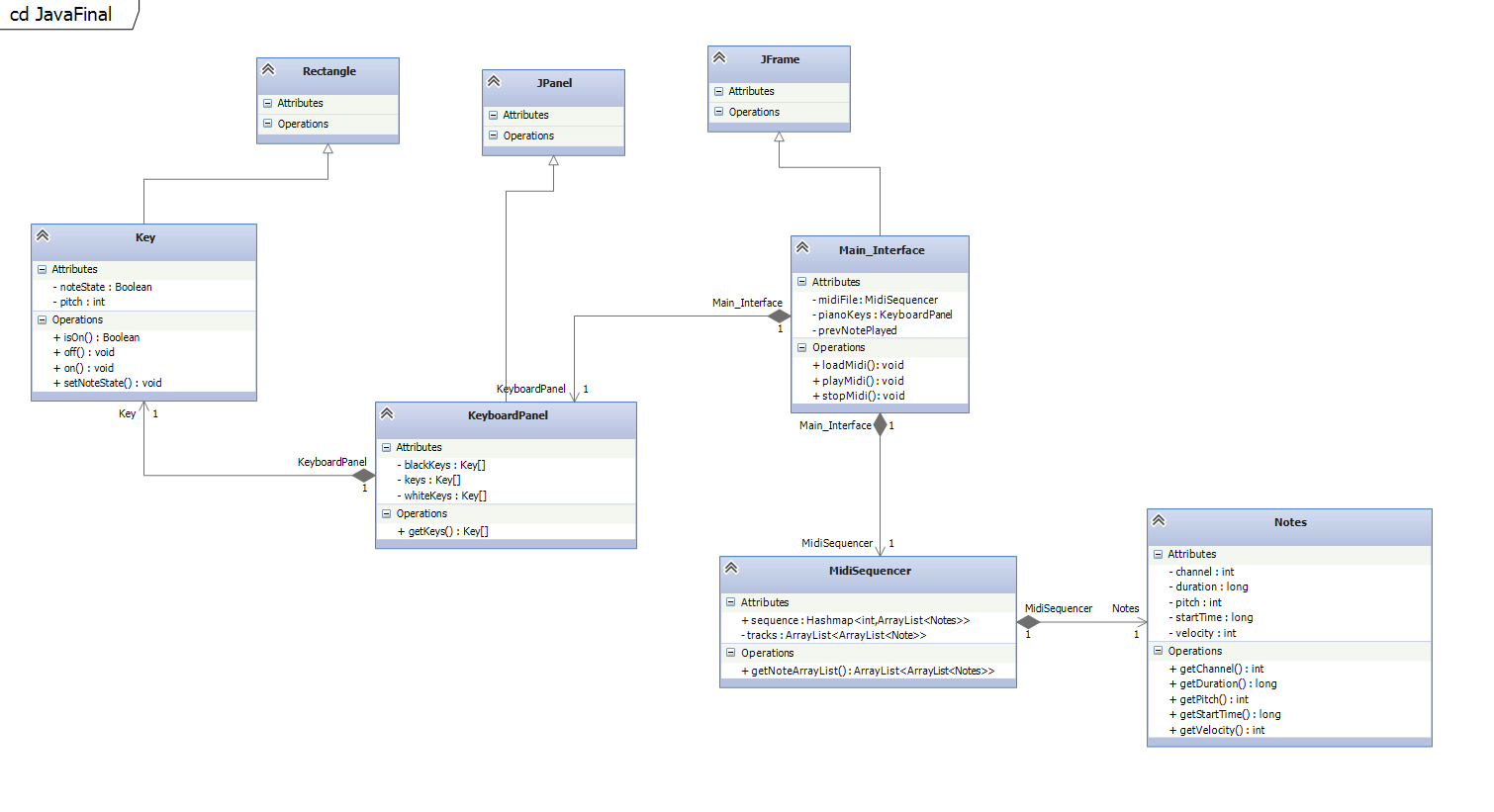
Virtual Piano and MIDI file player

* Project Functionality:
  + This code will create a virtual piano GUI that will play piano tones based on which key the user hovers over. An animation will occur for each key graphic that is played to display that it is currently in an action that plays a tone. The code will also have a function to load and play a MIDI file given by a user through a system browser. When a file is selected and the play button is pressed. The virtual piano will start playing the MIDI file through piano tones and press and animate the corresponding keys on the piano. Users should have the ability to start and stop MIDI file and resume playing at any given time.
  + Extra function if time is available:
    - Animate a graphic that displays incoming notes by a scrolling graphic of rectangles in the same x-axis as their corresponding note keys and their duration as the rectangles’ length. The piano will then play the key when a rectangle hits its corresponding key. (Think guitar hero or Synthesia)
* Classes
  + Main\_Interface extends JFrame
    - Uses:
      * MidiSequencer, Notes, KeyboardPanel, and Key classes
    - Function:
      * Creates the JFrame in main. Will house the functions to load (Create a new MidiSequencer), play, and pause the current midi file being played.
      * Sets up the interface with the KeyboardPanel class in a borderLayout
      * Will hold a thread to keep track of the current time to be able to start and stop a midi player
  + KeyboardPanel extends JPanel
    - Uses:
      * Key class
    - Function:
      * Creates a visual of Key objects(Rectangle objects) by loading all the white key pitches and black key pitches when creating Key objects with pitch integer values as a parameter during creation. The white and black keys will be stored in different Key Array so that the constructor of the KeyboardPanel can dynamically space and position white and black keys according to their pitch values. Then, it will add the keys from both arrays into a single array, keys, with indexes corresponding to the pitches of the notes (Highest pitch will be index 87, Lowest pitch will be index 0).
      * A mouseMotionListener will then be added to call a play function of the key the user hovers over on the GUI. This will check if a new key is hovered over and if so, plays the corresponding key’s pitch value. As will, this will implement a receiver function for when the Main\_Interface selects play and reads through the MIDI notes, it can transmit an event to make a key on the keyboard be played. This would also animate the current key being played.
  + Key extends Rectangle
    - Function:
      * The class is used to represent the keys on a piano as individual rectangles that will hold a pitch value. It will also contain function that will turn on and off the note’s sound (on(), off()). This is so the playMidi() function in Main\_Interface can easily turn on and off notes based on the Note class’s start and duration variables.
  + MidiSequencer
    - Uses:
      * Notes Class
    - Function:
      * This class will use file input to take a MIDI file from a designation and convert it into an ArrayList<ArrayList<Notes>> which will be called tracks. Since, MIDI files have the capability to store more than one track, a two dimensional array list is needed. Then, the class will be able to convert the two dimensional ArrayList into a HashMap<int, ArrayList<Notes> > which the key is the starttime value (in Notes class) and the object is an ArrayList of the Note object from track that have the same start values. This is so that the playMidi function can easily access if there are notes on the current time and how many notes are there to be played. This would solve the issue of having to search through multiple tracks to try and find a specific note that starts at this specific time.
  + Notes
    - Function:
      * Stores the values read in by a MIDI file including the channel, duration, pitch, startTime, and velocity of a note event. This is used to identify the attributes once a note is played.



Changes and Experience

When I was working on the MIDI Sequencer class, I used reference code to figure out the structure of MIDI files and how to add them into an ArrayList. I found out that my reference code read through an older version of MIDI files. Apparently there are two structures that can produce ON/OFF notes in a midi file, having the same note again but with velocity 0, of a hexadecimal value at the beginning indicating when a note is on and off. Since a majority more use the ON/OFF hexadecimal values, I had to find a new reference for code to read the files. My goal was to pass the Array Lists of notes to my main program. How original arraylist separated all the notes by the track value in the MIDI. I changed the track value to be the channel value since I wasn’t using more than one instrument (channel in a midi file is the type of instrument being played. I then pushed everything into a HashMap and used the start time of a note as it’s key. This was so that it would be much faster accessing a note at a specified time rather than iterating though and array. The HashMap stored the start time as the key and another arrayList of notes for the case of more than one note having the same start time.

In my KeyboardPanel class, I used a Hashmap as well to store the Key objects by their pitch. Using the Key’s pitch as the key for the hashmap also made it faster when trying to find the correct key to animate and be played. When researching how to play notes by a pitch, I had to include new objects that were unknown before such as Instrument, MidiChannel ,and Synthesizer. These were essential for playing the right notes given pitch and velocity. I also had to create vectors and new Arraylists that stored the notes to be able to access, remove, and animate Key objects while multithreading (Had issues with iterator before). I also had to implement a listener to find the point of the mouse and play a corresponding key in the JPanel KeyboardPanel. To do this I had to implement point and a action listener to KeyboardPanel itself and play and repaint a note if the mouse is hovered over it. An issue I ran into was since a black key and white key can occupy the same x,y corrdinates, I had to make it so that only the black key plays in that instance.

In the main interface, I had to add buttons to load and play a midi file since I wanted to do the extra functionality mentioned at the start of the project :

* + - Animate a graphic that displays incoming notes by a scrolling graphic of rectangles in the same x-axis as their corresponding note keys and their duration as the rectangles’ length. The piano will then play the key when a rectangle hits its corresponding key. (Think guitar hero or Synthesia)

For this I had to make a FileDialogue object to take a selected file path and check if it’s a .mid file to read in. only then can you be able to press the play button. I also had to implement a thread in the Interface class so that when play is pressed, the graphics can be played. A lot of it was repainting and setting up new classes like AnimationPanel and AnimationRectangle which produced the bars scrolling down the screen. Like before, I used HashMaps and ArrayList data structures to efficiently read and update the panel (Since music is timed based, I had to make it as efficient as I could so the song wouldn’t lag behind; Had issues with lagging when importing MIDI files with 80,000 + notes.). The thread will find all instances of Notes with the corresponding time is a HashMap and add it to AnimationPanel. Then AnimationPanel when being repainted will find all current AnimationRectangles, display them, and if finished, delete them. Lastly, I tried animating the keys being played with the rectangles with they hit their corresponding key by doing a check if y < 780 (y of the keys). I wanted to animate the color too but time ran out.

Overall I had a great experience coding this java project. I would say the only other additions I would make would be to animate the colors on the keyboard and also optimize the code for faster processing because now the play speed is limited to how fast it can process the data (which is why midi files with 80,000+ notes take forever to load/process and animate on the screen). Since java has everything stored as a reference and the garbage collection only deletes things that have no more references, I found it hard to optimize a lot of the reading through a loop function since I can’t delete an object directly like in C++. The most challenging area in my code was using a thread to animate the rectangles scrolling down the screen since it had to access the currentRectangles HashMap and ArrayList while its being updated with new rectangles being created and old ones being deleted. This is also why speed and optimization is key (it would through a lot of nullpointerExceptions and stop the graphics from animating since animation lagged behind and tried to animate objects that are already deleted). The cool factor in the end was just being able to visualize incoming notes like in guitar hero, it makes you appreciate how complex music structures can be and how cool it looks as a stream. I think this code is something worth showing peers and sharing since the other midi file stream players I used to reference cost money (FL Studio and Synthesia).

New Classes

* AnimationPanel extends JPanel
  + Functionality
    - Houses the panel where all note key rectangles will be held.
    - Keeps track of new and old notes(Animation Rectangles) currently being played and deletes notes that are done.
    - Draw and color the current notes and find the corresponding key on the keyboard and set it’s state to true or false
* AnimationRectangles extends Rectangle
  + Functionality
    - Creates a rectangle object with x and y coordinates based on the note read in by the MIDISequencer.
    - Uses pitch to find a Key objects x-pos and sets it as it’s own.
    - Y length is deteremind by the duration of a note in MIDISequencer
    - Animates by increasing the y corrdinates for every time the thread loops in the main interface.

