MooScore Project Report

Ben Young, Rahul Prabhu

Introduction

Writing sheet music by hand is a very laborious task, so most music writing is now done with a computer program. These programs allow the user to edit the sheet, playback their work using synthesized instruments and finally export the finished project to a PDF or other printable document. These programs allow users to implement the complete set of extensive music notation and create distributable final products. This complexity means however that these programs have steep learning curves, and some have steep price tags or limited functionality for free versions.

Literature Review/Background Study

There are several prominent music writing programs currently available, including Musescore, Finale, and Sibelius. These programs are heavy-weight, commercial products designed mainly for professionals and experienced writers. The goal for this project is to create a lightweight simple to open and use program that will allow for editing in basic music notation, playback, and saving/loading music. We will call this program MooScore, a completely original name that is an homage to our experience as UC Davis students.

Methodology

We will be writing our program with the Java Swing framework, which is a component-based GUI library for the Java programming language. The Swing framework will allow us to quickly iterate on our design and make the program cross-platform compatible. Our first step in designing our app was creating a layout sketch, and deciding which features we wanted to implement. We are musicians as well as programmers, so it was easy for us to understand how music notation should function and which features would be essential and which are optional.

After creating a sketch for the layout, we designed the program's UML diagram which would describe what components we would need to implement for our program. Creating each component from the bottom of the dependency hierarchy upwards allows us to test our progam as it becomes more complex, and identify missing/necessary features that will need to be implemented.

While writing our program, we were able to largely maintain our planned architecture, however some parts of the control logic were difficult to implement because of control needing to flow between different components. In order to maintain encapsulation, the main GUI component is passed as an argument while constructing the supporting GUI component so that they can call methods within the main GUI without having access to it's full state.

Java swing is designed to take advantage of several prominent programming patterns, namely Encapsulation and Observers. Each component within a Swing application is designed to contain all relevant data regarding its control and display characteristics. We take advatantage of the built in Swing capabilities and extend it by using inheritance and method overriding. Overriding the paintComponent method allows us to control precisely the graphics displayed on the screen, while Swing takes care of the low level implementation details. Event listensers, which are a flavor of the Observer pattern, are used frequently in our project to enable mouse and keyboard interaction, as well as synchronize components so that they display the correct information.

Implementation Details

Mouse interaction:
To enable dragging symbols around the screen with the mouse, a mouse handler is added to the GUI. This mouse handler, and all handlers within our project are an example of the Observer pattern, as well as the handler pattern. The handler extends an abstract class, but implements each required function as just an empty void function. Then, when we extend the handler, we can choose which classes we want to override without having to implement any of the functions we don't plan to use. The mouse event handler performs several functions within our Program. When a symbol is selected from the toolbar, it becomes active and will be shown at the cursor, and when a click is detected the symbol will be placed at the mouse's location. While no symbol is active, the mouse can be used to select and modify already placed symbols. If a component has been clicked on, it will so select it. The mouse event handler also uses mouse dragging events to compute and draw a rectangle from the start point to the end point of a user's mouse click to enable the selection of multiple components simultaneously. Selected components are shown in a highlighted color to help the user navigate the interface.

Playback:
Audio playback for our program is implemented using Java's MIDI library, which contains built in functions and objects that allow us to translate a sequence of notes from our GUI into a series of MIDI events, known as a track within the MIDI library. Notes duration is determined by their symbol, and their pitch is determined by their vertical position within their row. These tracks can be sent to a MIDI sequencer for playback, and are played at the user defined tempo. The MIDI library is powerful and may allow us to have additional voices, and choosing the instrument for playback.

Music Symbol Images:
To display the large number of different musical symbols inside our program, we use a large PNG file containing all of the symbols we need layed out in a grid. The master PNG, as well as the resulting smaller symbols, are examples of the Singleton pattern. They are eagerly initialized at the start of the program, but are otherwise static through the runtime of the application. By frontloading the expensive image manipulation calculations, there is minimal lag on user actions because the relevant images for the program are already stored in memory and are quickly accessible. Each symbol's image is extracted from the master image and is contained within an enum. This enum also contains information about the symbols height, width, preferred drawing scale, and the duration of the note if applicable. This information informs the GUI on how a note should be drawn, so that they can have the proper sizing as

well as maintain their height and width ratio so that they don't appear stretched or squashed. The note duration information is used to translate the GUI components into a MIDI track for playback. This system allows us to have minimal files within the source as well as a consistent visual style for the entire program. It would also potentially allow for several different symbol fonts to be selected from the user, such as a jazz font.

Text Input:
For user text input (titles, subtitles, composer, tempo), Jlabels and JTextFields were used. The GUI uses absolute positioning to place components onto the panel, so setting bounds for width, height, x, and y positions along with the current window height and width were used to place these components onto the screen dynamically.

ToolBar / MenuBar:
These components were extended the library components JToolBar and JMenuBar. By extending from these components, we were able to grab the default look of these components and then add the extra buttons that were specific to our project. Most of the buttons employed JButton, but for the notes specifically, the JButtons were created with Icons containing the desired resized note images. These note images are created using the same information as for the rest of the GUI so that there is a consistent look and feel.

Testing and Evaluation

Week 1:

During the initial week of planning and design, we decided to explore note placement onto the staff. We decided to pursue an approach that takes images of music notes and place them where the user clicks. Upon figuring out how to program event listeners that look for a mouse click and place an image, we determined that the centers of the images are not where the base of the note is, and that we could either create images that either fulfill this requirement or place images that are offset from where the user clicks to make it appear.

Week 2:

During Week 2, we refined the features that we had implemented last week. Since we had prototyped the ability to add/move notes and generate the staff / measure lines the previous week, we worked on improving these features. Now, the staff lines are generated dynamically according to the location of notes: notes on the bottommost staff will generate a new one below if the user needs to add more notes. Notes within the GUI are sorted based on their vertical position, simplified to which row of the staff they belong to, as well as their X position on the screen. This will allow us to playback the notes from top to bottom, left to right. The measure tracking system was also improved so that when all the notes in a measure are added up, it equals 4 beats. The position of notes are adjusted to align with the measure bar locations and ensure no invalid measure state exists. Last week, the toolbar functionality was not implemented in the backend, so now clicking a note symbol in the toolbar sets the active note that the user wants to place. Visually, the title, subtile, tempo, and composer information was added to the GUI for the

users to customize. Upon customization, this data would get saved to variables for later use in the MIDI player and for file saving.

Week 3:

During Week 3, we worked on features for the MIDI Player, especially converting the notes that the user placed on the GUI into music notes with durations and pitches. After adding a visual slider and play button to the bottom of the GUI, we started working on sound functionality. By using the previous weeks work on playback, we had to use the sorted notes and convert their position values into pitches that the MIDI player would be able to use to create sound. Iterating over each note, we determined that we would center our MIDI sequence around the note middle C, which is represented by the value 60. By determining the position of all notes higher and lower than C, we could convert them into their respective pitches. We also had to consider the difference in pitches between two consecutive notes. For example, the difference in pitch between C and D is 2 (half steps), but the difference in pitch between E and F is 1 (half step). By creating an array of half steps from C along with some modulo / division operations, we were able to convert distance from middle C visually to a pitch that could be input into the MIDI player. We determined that future tasks would deal with implementing accidentals, advanced playback, and saving / loading the music into a file.

Week 4:

For the final week, we continued to refine the MIDI playback functionality. This included some bugs in tempo and note duration when we were adding notes to the sequencer. Also, some notes would have wrong pitches if they were too low or too high, so we had to add extra logic check the current Y position and determine which staff line it was on and subsequently what pitch it was. The playback slider code was also updated so that users could add notes during playback if needed. Additional testing and code was required for different use cases, such as pausing in the middle of playback, resetting the player and slider when the music was finished, threading for the slider…etc. Note placement code also had to be updated, since the rests and whole notes were being placed higher than where the user was clicking. Before, users would have to drag the notes down lower in order to place them where they wanted. This would also cause errors in the pitch determination of notes since we used note position to determine pitch. We added code to play nothing when a rest note was detected, so now users could place all the available notes from the toolbar. Accidental signs such as flat and sharp were also added to the toolbar so users could make a note flat or sharp. Finally, the menu bar was updated to save and load files. When the user saves a file, the application iterates through all the components on the GUI panel and saves it to a file labeled using the title of the current music sheet. The load file button does the opposite and reads the components from the file then paints them onto the GUI screen. We also repurposed the edit button options to select all, which replaced the shift function. The select all and clear button were then implemented.

User Manual

Upon opening the app, users are able to see a representation of a blank music sheet. Users will be able to immediately edit the music sheet with the preset notation attributes. Other than adding music, the user will be able to type in the title, subtitle, and author of the music. Some permanent settings for this version of MooScore are a treble clef, key signature of C, and 4/4 time signature. Users will also be able to enter information about the music: a title, subtitle, tempo, and composer. There are three other sections in the GUI: the menu bar, the tool bar, and the playback section.

User Story: Composing a Song

Once the user has opened the app, they will see the notes and rests they are able to place along the left edge of the screen and a blank score with text fields inside the main panel. They can begin placing notes by left clicking on the button for the note they would like to add, then left clicking inside the main panel. If they would like to place multiple of the same note, they can hold shift while clicking to keep the active note. Once placed, notes can be selected by either clicking directly on them or dragging a box around several notes. Selected notes can be moved either by dragging with the mouse, or by using the *WASD* or *arrow keys* on the keyboard. While selected, notes can be deleted with the *Backspace* or *Delete* keys. They can be made flat, sharp, or natural using the bottom buttons on the toolbar or the *F, H,* and *N* keys.

The user can edit the *Title*, *subtitle*, *composer*, and *tempo* text fields by clicking on the text they would like to edit, then typing their changes. Clicking any blank space within the main panel will deselect any selected notes or text fields. Large changes can be made to the piece using the options in the *Edit* menu along the top menu bar.

When the user is ready to hear their new song, they can press the *Play* button along the bottom of the screen, and use the slider to control the playback. Once they are done working on their song, they can use the *Save* option inside the *File* menu bar, and load their work again later using the corresponding *Load* function.

The Menu Bar

The menu bar will perform extensive functions that are beyond the scope of the tool bar. For now, it will include three buttons: file, edit, and help. Upon clicking the file button, users will be prompted to either save or load their music. When saving music, the application will prompt the user to select a directory in their file system. Upon confirmation, the application will take the current music sheet and save the sheet information into a file for the user to store on their file system. The file will be named after the title of the music sheet with the file type of .dat (ex. “My Music.dat”). Conversely, the load option will allow the user to take a previously save music sheet file and load it into the application. Clicking the load button will again open the file directory for the user to navigate and find the file they had previously saved. On confirmation, the current music sheet will be replaced with the one loaded from the file. The edit button will include the ability to select one or more measures of music and perform operations to them, such as shifting or clearing. The edit button provides a popup which provides the “select all” and “clear” functions. Pressing “select all” will select all the notes on the screen, which will allow users to shift them in any direction. Pressing “clear” will delete all notes from the screen. The help button will create a pop-up that will display the contents of the user manual to provide any clarifications about the functionality.

The Tool Bar

The tool bar will provide all the music note options and functions. These will include choosing note and rests (sixteenth, eighth, quarter, half, whole). Users will also be able to add accidentals to notes. The user will have to select a note, then press the flat or sharp button which acts as a toggle for adding/removing flats and sharps. Users will note be able to add sharps or flats to music notes that already have the opposite accidental. The current functionality will overwrite the current accidental with the new one that the user has selected.

Playback Bar:

The playback bar allows the user to play the music that is currently on the GUI / sheet. The user can click play and will subsequently hear the music representation of their notes. During playback, the user can click the same button to stop the music as it is playing if they want to take a break, and then resume when they want to listen to the music again. Once all the music has been played, the play button resets, and the next time the user presses play, it will play from the start of the music sheet. If the user wants to skip to different portions of their music, they can use the playback slider to iterate through the notes until they reach the part that they want to listen to. When they drag the slider to a certain position, the music is halted and only starts when the user clicks play again.

Keyboard Controls:

The user can control many parts of the main GUI using the keyboard. The keyboard commands are as follows:

Number keys 1-5: Make the corresponding note from the toolbar the active note, 1 being whole note and 5 being sixteenth note. If shift is held down, the rest version will be activated instead.

WASD/Arrow Keys: Move the currently selected notes in the corresponding direction

F, G, H, N: The user can change the accidental of the selected notes. *F* for flat, *G* for none, *H* for sharp, and *N* for natural.

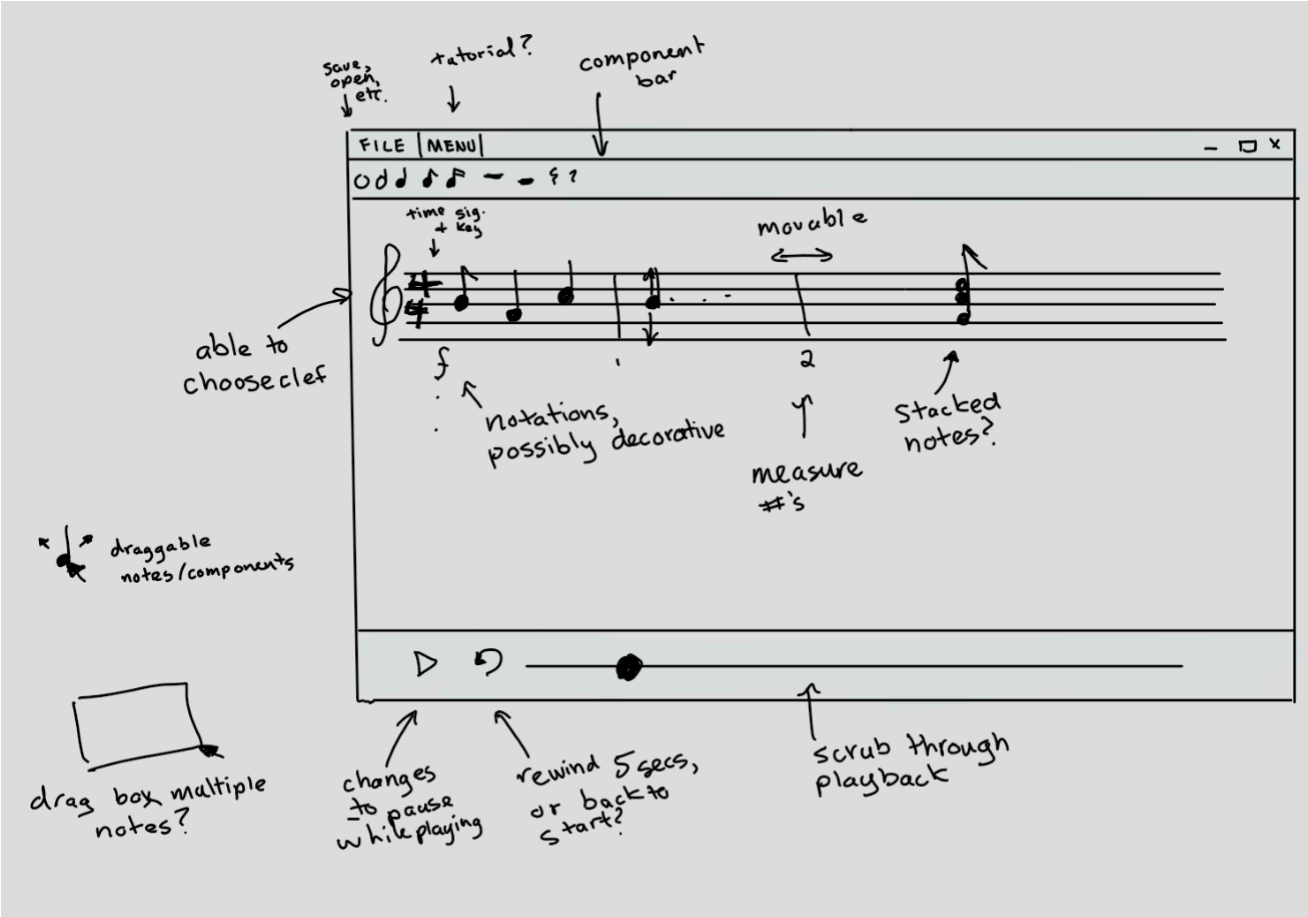
Delete/Backspace: The user can delete notes they have placed using either *Delete* or *Backspace.*

Escape: This will clear the active note, that is if the user selected a note to place and would like to clear that selection they can press escape.

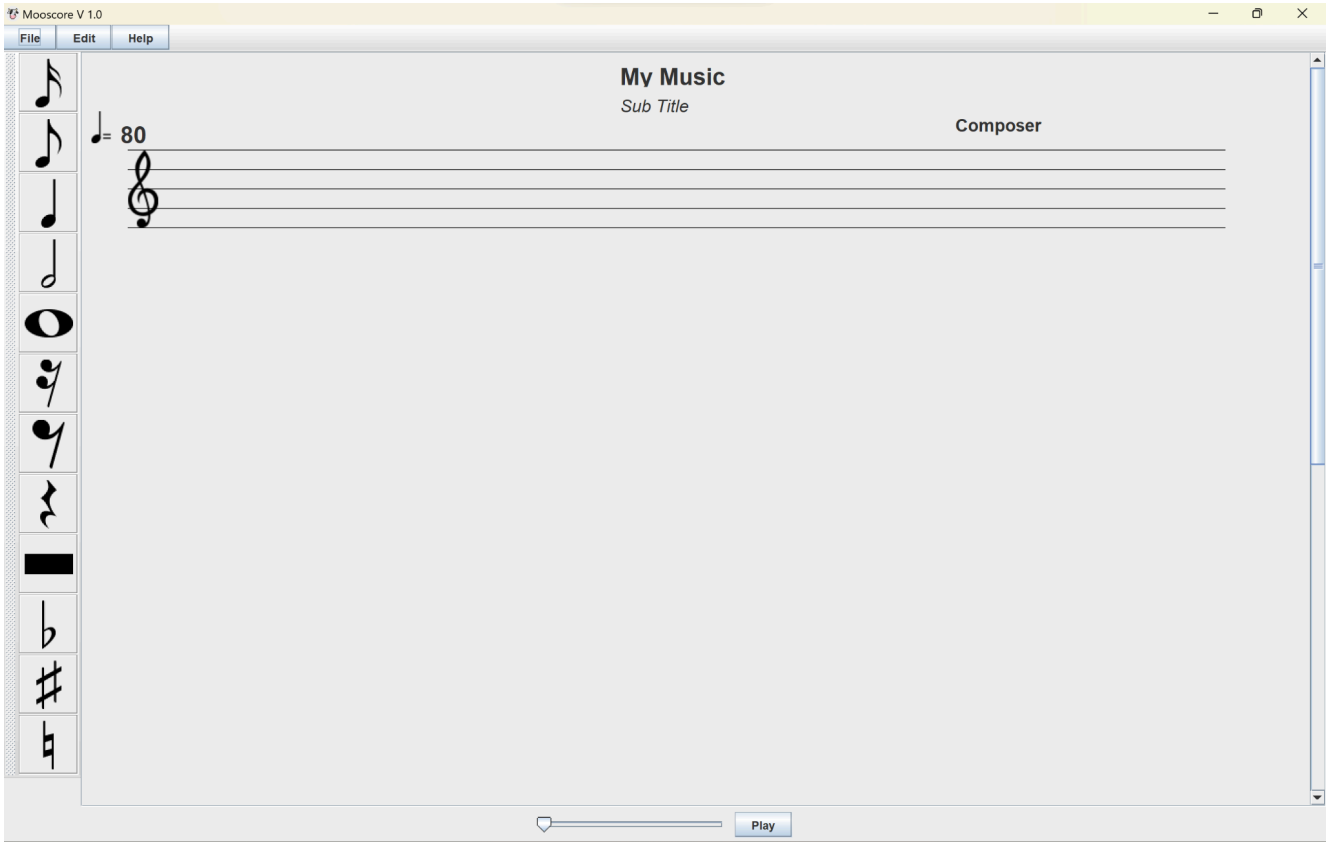
Pressing any key not listed above will deselect all currently selected notes.

Design Manual

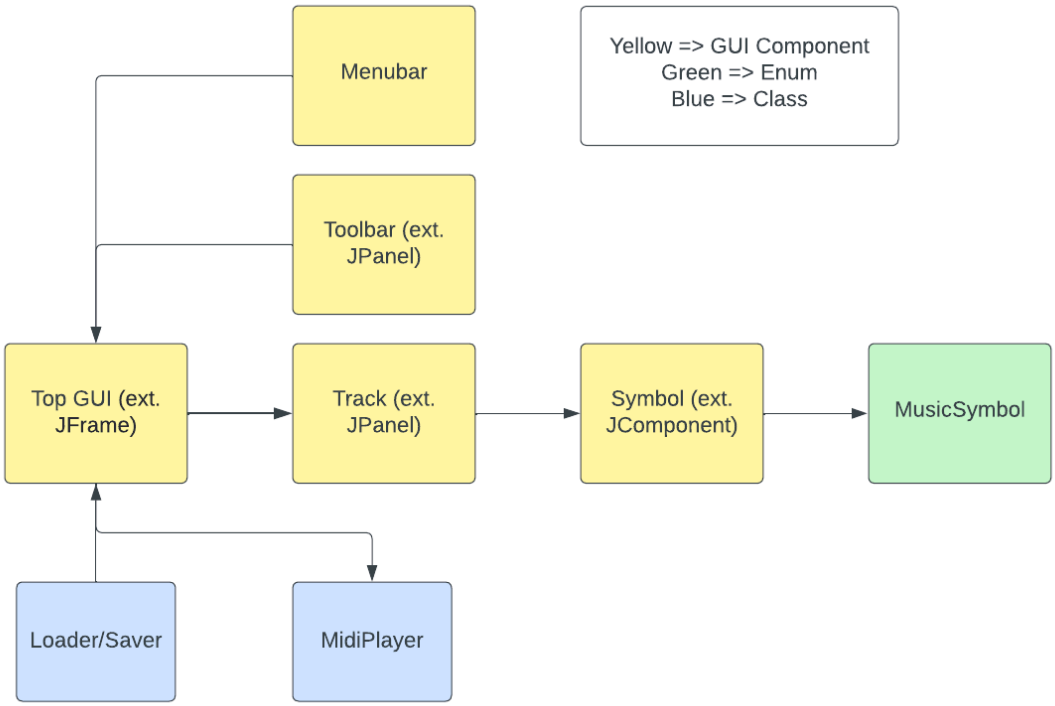
Initial Layout Sketch:



Final Layout:



Class Diagram:



Appendix

Chat GPT Logs

- Chat GPT Log - Ben Young
    - Button Size with ImageIcon: Setting the size of a JButton constructed with an ImageIcon. The preferred size can be set using button.setPreferredSize(new Dimension(width, height)), but if it doesn't seem to apply the new sizing, you might need to revalidate and repaint the container after setting the preferred size.
    - Maximizing a JFrame: Maximize a JFrame window in a Swing application using frame.setExtendedState(JFrame.MAXIMIZED_BOTH). This maximizes the frame both horizontally and vertically.
    - Checking if Two Java MIDI Sequences are Equal: Method to check if two Java MIDI sequences are equal by comparing their contents, including division type, resolution, tracks, and events.
    - Setting the Size of a JButton: Setting the size of a JButton using the setPreferredSize() method, either directly or when the button is constructed with an ImageIcon. However, if the preferred size doesn't apply, you may need to revalidate and repaint the container.
    - Custom Swing Component Alignment: We discussed aligning components in a custom JComponent, such as a JPanel, with absolute positioning by overriding the paintComponent() method to draw children components with custom positions.
- Chat GPT Log Week 1-3 - Rahul Prabhu
    - This log includes the ChatGPT prompts and answers from weeks 1-3 for Rahul Prabhu. The log includes questions on how to draw images such as music notes onto a Swing GUI, adding listeners for button and mouse interactions with drawn components, adding toolbars, menubars, textfields, labels, and resized images to the GUI. Some prompts included code which ChatGPT was able to optimize or revise to fix bugs. Further prompting was used to account for edge cases or bugs that ChatGPT didn’t consider. Specific prompts and summaries of ChatGPT answers are shown below.
    - **How to add a background image to a canvas in a Swing project?**
        - Provided code to load and display a background image on a Swing canvas.
    - **How to draw a quarter note on a canvas using only paint?**
        - Provided code to draw a quarter note using the paintComponent method in Java Swing.
    - **How to adjust the code to add a background image to a canvas?**
        - Revised the code to include a background image on a Swing canvas.
    - **How to set the radius of the note and the stem thickness to fixed values?**
        - Adjusted the code to set fixed values for the note's radius and the stem's thickness.
    - **Is there a way to remove an image placed on a canvas by clicking on it?**
        - Suggested implementing mouse event handling to detect clicks on the canvas and remove images accordingly.
    - **Are there Java Swing methods to let me drag and drop images?**
        - Recommended using a more appropriate component for drag-and-drop functionality, such as JLabel.
    - **How to create a file, edit, and help button on a toolbar on top of a Swing app, and make it a component of another frame?**

        - Provided code to create a toolbar with buttons for file operations and positioning it within another frame.
    - **How to create a pop-up window with text when a user clicks a button?**
        - Demonstrated creating a pop-up window using JOptionPane.showMessageDialog().
    - **How to check the operating system in Java?**
        - Explained how to use System.getProperty("os.name") to retrieve the operating system name.
    - **JToolBar vs. JMenuBar?**
        - Provided a comparison between JToolBar and JMenuBar in terms of their purposes and usage.
    - **How to create a JButton with an icon?**
        - Provided code examples to create a JButton with an icon using ImageIcon and BufferedImage.
    - **Is there an alternate JButton that can be selected and deselected?**
        - Introduced JToggleButton as an alternate button that can be toggled between selected and deselected states.
    - **How to create a bigger JButton with a smaller icon inside?**
        - Provided code to create a larger JButton with a smaller icon inside, adjusting the button's size and icon size accordingly.
    - **How to add an ActionListener to a JButton that sets a variable to 1 when pressed?**
        - Demonstrated how to add an ActionListener to a JButton to set a variable to 1 when pressed, along with an example code snippet.
- Chat GPT Log Week 4 - Rahul Prabhu
    - This log includes the ChatGPT prompts and answer for week 4 for Rahul Prabhu. The log includes prompts to help create features and bugs that were worked on during week 4. Some concepts that were discussed were Java programming concepts, MIDI sequencing, complex actionlisteners / metaEventListeners, multithreading, and file handling.
    - Java Programming
        - Enums, file handling, exception handling, and static vs. non-static methods and fields.
    - **MIDI Sequencing:**
        - How to create MIDI sequences, tracks, and events using the javax.sound.midi package in Java.
        - How to add notes, rests, and control change events to MIDI tracks.
        - Tempo control and how to set tempo in a MIDI sequence.
    - **Swing GUI Development:**
        - Swing components, event handling, and how to create and manipulate GUI elements in Java.
        - Adding action listeners to buttons, handling user input, updating GUI components dynamically, and clearing components from a GUI.
    - **Multithreading:**
        - Threading concepts, including creating and managing threads, interrupting threads, and joining threads.
        - Updating GUI components asynchronously using a separate thread.
    - **File Handling:**
        - How to read from and write to files in Java, as well as how to handle file system operations.