Design Manual

Architecture Overview

The software follows a structured architecture, primarily centered around the main class MusicNotationEditorUI. This class serves as the core component, encompassing the entire application functionality, including the initialization of toolbars, creation of the music staff, and setting up the playback. The rest of the software is composed of separate classes that rely on Java Swing container classes. The majority of interactions are determined through right and left clicks of the mouse. The architecture employs a modular approach, allowing for the seamless integration of various components.

Design Patterns

Factory: Naively so, our design implements the factory design pattern. With an abstract class of MusicSymbol, later more thoroughly explained, we have the interface for creating the music symbols needed which can encapsulate notes, symbols, etc. It is then followed by the concrete classes with their need for the factory method clone().
State: The state design pattern is utilized in the playback to manage the state of the playback state. This would essentially allow transitions between playing and paused states seamlessly. This pattern enhances the flexibility of the playback functionality.
Template method: To prevent having to rewrite an entire class for each type of symbol, we implemented our simple idea of the template method design pattern. This involves the abstract class MusicSymbol and being able to let the concrete child classes have similar yet different common behaviors. This encourages some level of consistency and code reusability in our design.
Prototype: While this part can be touched as the factory design pattern, the prototype design pattern is also represented with the method clone(). This allows us to create new objects without specifying the class too much and promotes the reusability of the selected object.
Command: Like a remote control with its buttons, the command pattern was introduced to encapsulate playback behaviors–play and stop–as command objects. This pattern provides a clean way to handle the UI actions in our software. Concrete command classes (PlayCommand and StopCommand) were created, and their integration with the UI buttons was outlined and will be integrated.

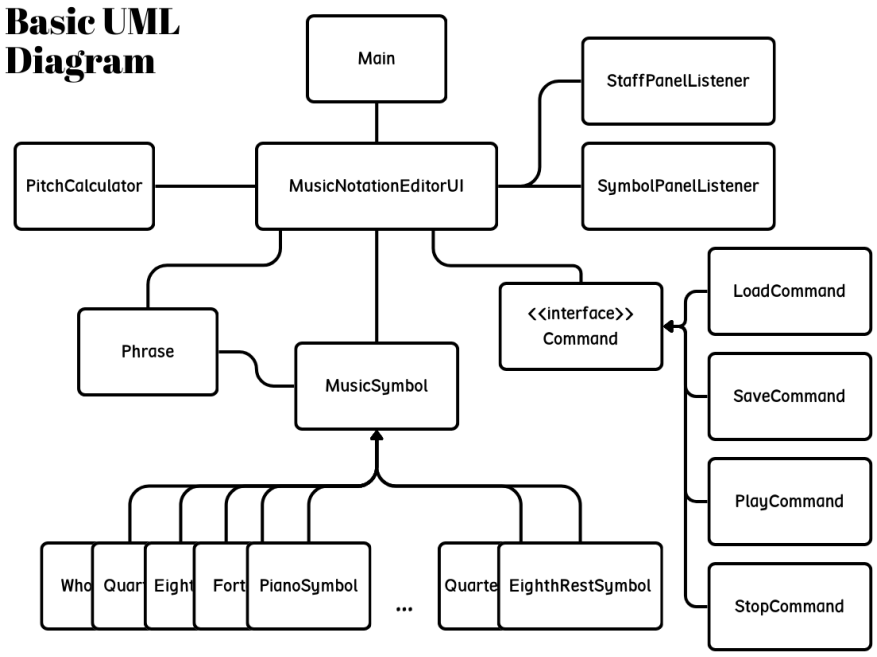
Component Descriptions

MusicNotationEditorUI - responsible for managing the entire application. It controls the music score, toolbars, and playback functionality. The MusicNotationEditorUI class facilitates interactions with other components and orchestrates the overall behavior of the application.
Playback - Manages MIDI playback, reading note data, and producing corresponding sounds. This class ensures synchronized playback according to the music score and handles transitions between different playback states. In the code, this is represented by the interactive buttons on the UI and the behavior that follows the interaction.

Toolbar - Creates and manages toolbars for adding musical elements, such as notes and rests. The Toolbar class provides a graphical interface for users to interact with various musical elements, facilitating music composition and editing. Currently, the toolbar visually includes the four different note styles we have provided and there are plans to expand the toolbar for more uses.
Phrase - The class Phrase is to keep track of the four measures. It is also in charge of tracking measures and managing the placement of notes and, hopefully in the future, sharps and flats as well. It offers functionality to add additional empty phrases for composing music notation. Due to time constraints, certain functionality was not fully implemented between Phrase and Measure. See below for more details on Measure.
MusicSymbol - Abstract class encompassing various musical elements, including notes, clefs, and key signatures. It ensures a unified approach to handling different musical components within the application. As to the fact, many of those symbols have/will have similar behaviors of being drawn out, being placed on the staff, etc. Currently, the implementation involves the use of some basic music notes.
Note - This class involves the creation of the actual note like drawing it onto the measures. It will create instances of each valid note (within the time signature constraints) and adjust the pitch accordingly (depending on the key signature as well). If, in case of a rest, it will still keep track of duration and just have no sound on the pitch. There will be more to implement to create different whole, half, quarter, and eighth notes/rests.
Dynamic - With the basic music dynamics, we are going to implement the ability for the user to select one of the few most common dynamics, piano, mezzo piano, mezzo forte, and forte. This would allow the user who is creating their music sheet to decide on the volume of the playback more flexibly and have a better, faster way of comparing the adjustable volumes by level as opposed to controlling the volume of their device.
Measure - The concept of a measure is implemented visually in the structure of our program. Our intended behavior of the Measure class was to allow the user to add measures as they wanted. This would limit that they could *only* put notes on the measures they have made and created, and have a cleaner interface as the user iterates through their music sheet. Adding more measures would eventually create a new phrase below. Due to time constraints, this full implementation was not initiated in the main submission of our project. Code for the Measure class can be found in this branch of our Git repository.
Time Signature - For now, the implementation of our time signature is simply a graphic on the staff panel. Because common time, four-four time, is the most common, we decided to just stick with that time signature for basic teaching using our software.

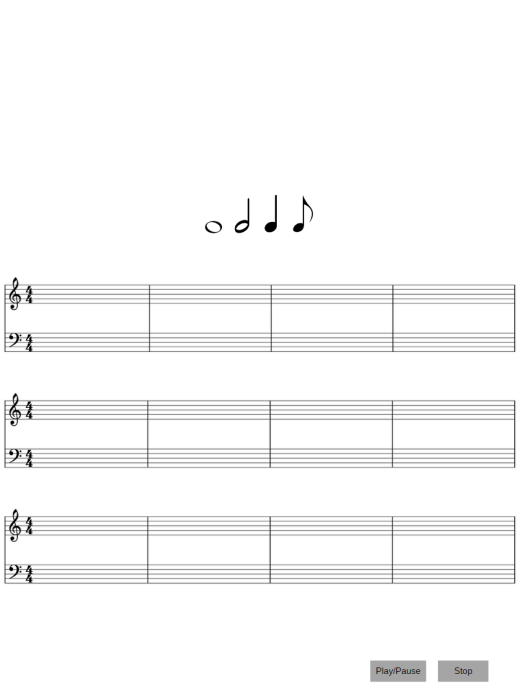
Note: Many of these aren't fully implemented yet such as the complete implementation of dynamic adding of measures, and adding more time signatures.

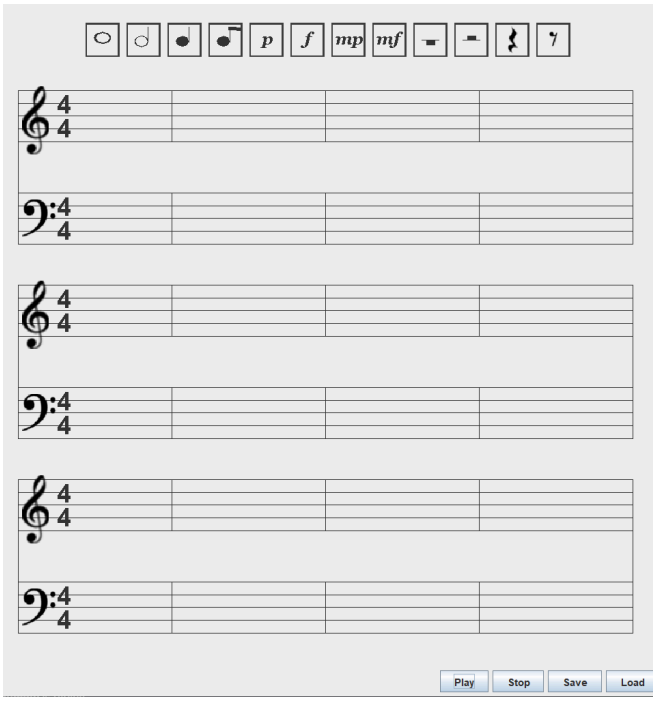
Diagrams and Basic Sketch



This is a UML class diagram representing the structure of the music notation application. It outlines the relationships and responsibilities of the various classes involved in the application. As the project progresses, the class diagram will evolve to reflect any changes in the architecture and design.

Original Basic Sketch & Current Look:





Standards and Conventions

Descriptive Variable Names: Variable names such as staffPanel, symbolPanel, etc., are chosen to indicate their purpose or functionality within the code.

Proper Indentation: The code follows consistent indentation practices, enhancing readability and making it easier to understand the structure and hierarchy of the code.

Meaningful Comments: Comments are used throughout the code to explain the purpose of methods, classes, and code blocks. For example, a description of the initialization of components, arrangement of components, and setup of action listeners. Other comments are temporary (most are generated by ChatGPT) and will be removed after refactoring.

Encapsulation: Encapsulate related functionality and data, promoting modularity and reusability. For example, the StaffPanel class encapsulates the logic for drawing staff lines, while the ControlPanel class encapsulates the logic for managing control buttons.

Listener Classes for Event Handling: Introduced separate listener classes (e.g., StaffPanelListener and SymbolPanelListener) for handling mouse events, promoting a clear separation of concerns. This design choice aligns with the Single Responsibility Principle, ensuring that each class is focused on a single aspect of the application's functionality.

Abstraction and Polymorphism: Leveraged through the MusicSymbol class hierarchy, enabling a unified approach to handling different types of musical symbols. This use of abstraction and polymorphism simplifies the management of diverse musical elements, enhancing the code's flexibility and scalability.

User Stories:
As a user, I should be able to see a graphical representation of a musical staff displayed on the editor interface.
As a user, I should be able to visually place notes by clicking on specific positions on the staff.
As a user, I should be able to edit existing notes by clicking on them and making changes to their properties, such as pitch and duration.
As a user, I should be able to playback the composed music to hear how it sounds, helping me evaluate and refine my compositions.
As a user, I want to be able to pause and resume playback at any time, allowing me to analyze specific sections of the composition more closely.
As a user, I want to have a toolbar or menu where I can select different note durations (whole note, half note, quarter note, eighth note) to add variety and complexity to my compositions.

BHAG: As music and tech enthusiasts we wish to create a comprehensive music notation editor that combines intuitive interfaces with powerful functionality, enabling musicians of different levels to create and playback music scores. By leveraging design patterns and strategies, our editor aims to set new standards in the field of music notation software, empowering users to unleash their creativity and bring their musical visions to life like never before.

Project Report

Introduction

This project aims to develop a simple music notation software tailored for educational purposes, catering to both students and teachers. The primary objective is to provide users with an intuitive platform for creating and editing music notation effortlessly.

Goals:

1. Educational: Create a basic music notation software suitable for beginner-level music education.
2. User-Centered Interface: Design a minimalistic and user-friendly interface to facilitate easy editing and navigation.
3. Replicating Basic Music: Develop software capable of composing and playing music sheets such as “Twinkle Twinkle Little Star,” “Happy Birthday,” and “Hot Cross Buns.”
4. Scalability: Ensure that the software architecture allows for future expansion and integration of advanced features to accommodate the evolving needs of users.

Scope: The project aims to deliver software with a fundamental feature set ideal for beginner music education. While the primary goal is to accommodate beginner-level music, the stretch goal includes the ability to compose music scores up to the preparatory A level of the Royal Conservatory of Music standards. Possibility to expand the project scope to include support for MIDI file import/export functionality, enabling users to work with existing musical compositions and collaborate with other software platforms.

Significance: Foster digital literacy in music education and its potential to inspire creativity and innovation in software development**.** Aside from the educational significance to our users, we want to hone our skills in software development. This involves understanding design patterns, and software development principles, and being able to apply those skills to the project that we are making from scratch. Additionally, mastery of Java Swing in GUI development is a key focus area.

Literature Review/Background Study

The project draws inspiration from existing music notation software such as MuseScore and Flat.io. Analyzing the features offered by these platforms assists in determining the essential functionalities for our project. Understanding their menu structures, mouse interactions, and keyboard shortcuts aids in optimizing the user experience for our simplified music notation software.

Methodology

The development process follows Agile methodology, emphasizing iterative changes and continuous improvement. We hope this development process helps us in this relatively short time frame. Utilizing Git facilitates efficient collaboration and version control among team members. Java Swing is employed for GUI development, adhering to the project guidelines and instructional framework.

Implementation Details

We will be using Java Swing

Key development phases:

1. Designing the software and prioritizing core features.
2. Developing major functionalities for music staff display and basic notes.
3. Enhancing note functionality to include various styles and rest notes.
4. Implementing playback features to ensure accurate pitch representation during playback.

Pitch Calculation:

We introduced a PitchCalculator strategy pattern to calculate the pitch of musical symbols based on their y-coordinate on the staff. This approach allows for flexibility and future extensions, such as different clefs or tuning systems.

MusicSymbol and Subclasses:

Adjustments were made to your MusicSymbol class and its subclasses (e.g., WholeNoteSymbol) to include properties for MIDI pitch and duration, ensuring that each symbol knows its musical value. We discussed how to instantiate these symbols with the correct type and duration, allowing for accurate playback.

Playback Mechanism:

A conceptual outline for a playback mechanism was provided, focusing on iterating over symbols in a Phrase and playing their sounds based on pitch and duration.
The detailed implementation of sound playback (e.g., using MIDI) was covered and work will be done to make it more robust.

Command Pattern for Playback Controls:

Adopted for managing UI actions such as play, pause, save, and load, encapsulating each action's invocation and execution. This pattern facilitates decoupling between the user interface and the logic behind these actions, enabling easier modifications and additions in the future.

Testing and Evaluation

Our music notation editor was tested through both automated print statement outputs and manual, exploratory methods. We focused on validating pitch accuracy and symbol positioning by dynamically adding, removing, and repositioning notes to uncover any edge cases. To mimic real-world use, we role-played as music students and teachers, and also gathered external feedback from peers, enhancing our understanding of the user experience.

There weren't any bugs found in our latest implementation. Some feedback was that clicking the toolbar and clicking the staff to place a note was not intuitive but it is a main requirement in the project and, hence, we kept it that way. The non-functional stop button was noted and identified as a feature yet to be implemented rather than a bug. Overall, the application met essential performance standards, providing a foundational set of features for basic music theory exploration.

We were able to test the application by drawing out actual music scores or transposing our music such as testing Twinkle Twinkle Little Star, Stay with Me by Sam Smith, Do-Re-Mi from the Sound of Music, and When I Was Your Man by Bruno Mars, some of which are included in our Git repository.

Results and Discussion

Our project achieved standard success, meeting all required objectives and incorporating additional features for enhanced usability. Notably, the save and load functionality was introduced to offer users the convenience of returning to unfinished compositions, a feature not originally requested but deemed essential. This addition, along with the integration of a wider range of musical symbols beyond the basic note types, allowed for more comprehensive experimentation within the software. To further enrich the musical notation experience, dynamics and rests were incorporated, reflecting their critical role in music theory by enabling users to denote silence and control the song's intensity. The decision to include three pairs of staff lines was made to provide ample space for users to explore and notate music effectively.

In our project, we strategically selected design patterns based on intuitive reasoning and their alignment with our project's goals:

Command Pattern: Chosen for its suitability in managing UI button actions, offering a clear separation between command initiation and execution, akin to remote control logic.

Prototype Pattern: Applied to efficiently clone MusicSymbol objects for placement in compositions, proving essential for replicating complex objects with varied attributes.

Template Method Pattern: Utilized for its practicality in creating diverse objects through a consistent process, simplifying the instantiation of various music symbols.

We consciously decided against certain patterns due to the unique nature of musical symbols:

Decorator Pattern: This was considered impractical due to the extensive variety of symbols (notes, dynamics, rests), which complicated the application of a uniform decoration approach.

Composite Pattern: Not implemented as the heterogeneity of musical symbols and their grouping did not lend itself well to a composite structure, favoring a more straightforward organization method.

Our project, fueled by enthusiasm, considered the implementation of additional design patterns to enhance its architecture and functionality. However, time constraints led us to prioritize core features, leaving room for future enhancements. Initially, we contemplated using the Singleton pattern to manage the music score centrally, providing a singular point of reference for the entire project. This approach would have streamlined access and manipulation of the score across different components. Instead, we opted for decentralized control, with each Phrase object managing its notes and rests independently. This decision, while effective, diverged from the original plan to centralize score management. Another consideration was the Observer pattern, aimed at optimizing how changes to the music symbols—such as additions, deletions, and repositioning—are tracked and reflected across the application. Our implementation involved direct modifications to the ArrayList of MusicSymbols within each Phrase object, without employing traditional Observer pattern methods like update() or subscribe(). This choice was made to maintain progress but left the door open for future refinement to achieve a more decoupled and reactive update mechanism. With additional time and resources, we believe these enhancements could significantly improve the application.

We initiated the Command design pattern, notably for UI commands, but the Stop button functionality remains incomplete, highlighting a key area for future work. There was also the Measure implementation as previously mentioned in this branch. Efforts to implement dynamic measure addition faced challenges, leading us to continue with a static staff panel. It would be a great option to let the users add measures until they wish. Prioritizing the development of core components instead, at the same time duration, we created a staff panel, control panel, command panel, basic notes, and their main functionalities. It was difficult to teach ChatGPT what was wrong. Future updates will aim to integrate the Stop functionality and explore dynamic measure addition, enhancing the application's usability and feature set.

While brainstorming enhancements to deepen the "music theory" aspect of our project, we contemplated adding key signatures and expanding note options on the staff. These features, aimed at showcasing our music knowledge, were ultimately deprioritized to focus on demonstrating our software development skills. Implementing key signature options would improve the application by allowing compositions beyond the C major and A minor scales, thus serving as an educational tool for music theory. However, this was set aside for features more directly showcasing our design and development choices. More note options, including the ability to notate higher and lower pitches on the treble and bass clefs with appropriate ledger lines, were considered to offer a comprehensive notation range. This idea was balanced against the current scope, deemed sufficient for beginner piano students. Ideas for representing dotted

notes, beamed eighth notes, and staccatos were also explored to enhance notation capabilities, further contributing to the educational value of the program.

To enhance both user and developer experiences, we've identified several areas for future improvement, given more time and practice. It seems to us that it is a lot harder to organize the code the way we want it to be especially when ChatGPT has a hard time understanding that we only want up to 5 lines of code and/or little to no parameters but also try to keep the objects in our code safely encapsulated from each other. Visually, the application could benefit from higher quality note representations and PNGs, which, although not immediate priorities, would significantly improve its aesthetic appeal and usability. Additionally, MIDI playback has shown inconsistencies, such as lag and irregular beat timing (i.e. the 5th and 6th beat of each phrase was, for some reason, faster than the other beats) and the flow of the sound was not smooth despite the code being very straightforward. This would need more time to learn the MIDI library extensively. Introducing a snapping feature for note placement could also markedly enhance user interaction by minimizing placement errors and clarifying the musical score's visual and functional presentation. With a focused effort on these areas, including a more nuanced collaboration with ChatGPT for code optimization, we anticipate substantial advancements in the project's overall quality and user satisfaction in subsequent updates.

Conclusion

Throughout this project, we set out to develop a Java Swing-based music notation editor, aimed at assisting music students and educators in creating and editing basic musical scores. Our journey led to the successful implementation of core functionalities, including the ability to place, move, and remove notes on a staff, alongside basic playback features, contributing to a foundational tool for music theory exploration.

Achievements:

We introduced a range of musical symbols, dynamics, and rests, enriching the application's educational value.
The Command design pattern was effectively utilized to manage UI actions, enhancing the application's modularity and flexibility.
Serialization was employed to implement save and load functionalities, providing users with the ability to preserve and revisit their compositions.
Through intuitive UI design and thoughtful feature selection, we focused on creating a user-friendly experience while maintaining a solid architectural foundation.

Learnings:

Our iterative approach to development and problem-solving, supported by tools like design patterns, underscored the importance of adaptable and maintainable code structures.
Feedback from user testing was invaluable, revealing insights into usability challenges and opportunities for improvement, particularly regarding intuitive interaction and visual representation.
The exploration of advanced features such as dynamic measure addition and MIDI performance optimization highlighted areas for future growth and development. While we achieved significant

milestones, the journey highlighted the balance between ambition and practicality, pushing us to prioritize core functionalities over extended features.

Looking ahead, we are inspired by the potential for further enhancements, driven by our foundational achievements and the constructive feedback received. Our vision for the future includes refining the UI, optimizing MIDI playback, and expanding the application's music theory capabilities. This project has not only contributed a valuable tool for music education but also provided us with profound insights into software development practices, teamwork, and the iterative nature of creative problem-solving.

In the end, this project was a big step for us. It's both an achievement and a starting point for making even better tools for learning music. We're excited about what we've learned and what we can do next.

User Manual

Introduction

Welcome to the Music Notation Editor user manual! This guide is designed to help you effectively utilize the features and functionalities of our music notation software. No matter your proficiency level, this manual will provide you with step-by-step instructions and guidance on how to create and edit simple musical scores.

Installation Instructions

To install the Simple Music Notation Editor on your computer, follow these steps:

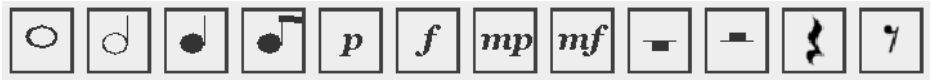
1. Download the installation file from <xyz>.
2. Run the installer and follow the on-screen instructions to complete the installation process.
3. Once installed, launch the application by double-clicking the icon on your desktop or from the Start menu.

Overview of Features

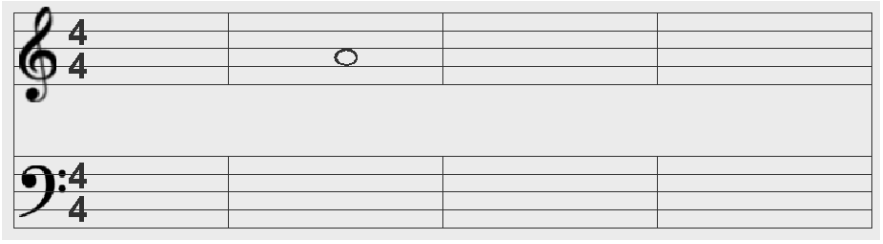
- Staff Display: View and edit musical scores on a graphical representation of a musical staff.
- Note Placement and Editing: Easily add, move, and delete musical notes and symbols.
- Playback Functionality: Play back your compositions to hear how they sound in real time.
- Note Duration Selection: Choose from various note durations (whole note, half note, quarter note, eighth note) to create rhythmic patterns.
- Sheet Dynamic Selection: Choose a single dynamic to control the volume of the music.

How to Create and Play a Music Score

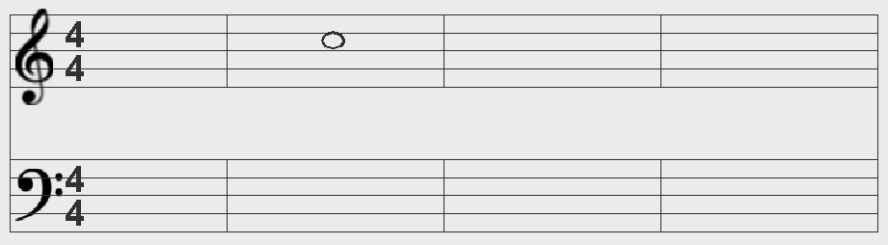
1. With the toolbar, select the music symbol you wish to add with a left-click



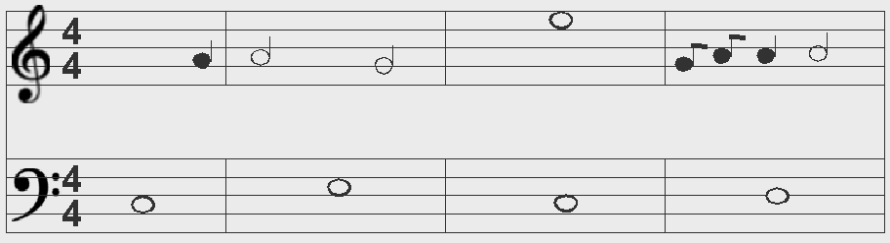
2. Place it down onto the staff panel with another left click.



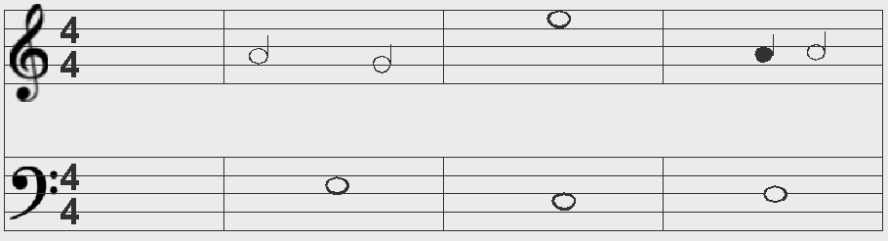
3. To move a note, click and drag it to the new desired position on the staff.



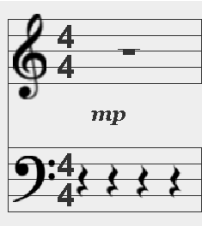
4. Repeat the previous steps to create a sequence of various notes accordingly.



5. To delete a symbol, right-click on the symbol.



6. Other symbols on the symbol bar can be interacted with in the same way. If you wish to add a dynamic for a little more advancement, select the wanted dynamic and place it at the beginning of the music score. Adjust around if you wish to use rests.



7. Click on the “Play” button to start the playback of your composition.

8. Click on the “Save” and “Load” buttons to save the sheet or load a previously saved file.

Troubleshooting

If you encounter any issues while using the Simple Music Notation Editor, try the following troubleshooting steps:

- Click with patience, adding too many notes at once will complicate the process.
- Ensure that your computer meets the minimum system requirements for the software.
- Restart the program and check user permissions match.
- Check for updates to the software and install any available updates.
- If the issue persists, contact our customer support team for assistance.

FAQs

Q: Can I import MIDI files into the Simple Music Notation Editor?
A: MIDI file import functionality is not currently supported in the software. However, it may be considered for future updates.

Q: How do I delete a note?
A: Right-click on the note symbol to delete it.

Q: Is technical support available?
A: Yes, for any further assistance or technical support, please contact our customer support team.

Q: How do I change the key signature of my composition?
A: The key signature can be changed by selecting the desired key signature from the toolbar and placing it on the staff panel.

Q: How do I add more dynamics across the music score?
A: This simple music notation editor is structured for the very basic. So there is no option to have multiple dynamics. To change the dynamic you have selected, remove the previous one on the sheet and place a new one.

Q: I tried to add a note but instead it dragged a note beside it to that location. Where is the new note I wanted?
A: Be careful with the spacing of your notes! If you configure notes too closely and without good notation practice, the notes may overlap with each other and you would have to drag them apart.

Appendix

File Name: ChatGPT_Logs_for_Sohum.html
Brief Summary:

- Project Description and Timeline: The conversation began with a discussion of a Java Swing project aimed at creating a Simple Music Notation Editor. The project was broken down into phases, with Week 1 focusing on designing the user interface (UI) and planning the implementation of core features.
- Design Manual Specifications: The user provided specifications for the Design Manual, detailing its focus on the software's architecture, design patterns, component descriptions, diagrams, and coding standards.
- Implementation of UI: The user requested implementation of the UI in Java Swing, including staff lines, control buttons (play/pause, stop), and symbols for musical notes. They also emphasized the need for appropriate Java design patterns and good OOP practices.
- Refinement of Code: The user requested refinements to the Java code, including fixing repetition, implementing graphical representation for music symbols, and enabling click-and-drag functionality to place symbols on the staff.
- Organization and Code Separation: The user asked for further organization of the code into separate Java files to improve readability and maintainability.
- Drafting the Design Manual: The user provided a draft of the Design Manual and requested improvements and reorganization to ensure compliance with the provided specifications.Brief
- Additions to Design Manual Sections: Additional points were added briefly to each section of the Design Manual to enhance clarity and completeness.
- Discussion on Testing and Evaluation: The user reminded that testing, evaluation, results, and conclusion sections were not yet part of Week 1 deliverables but discussed their potential execution in future phases.
- User Story Mapping: The user requested assistance in creating user stories for the features of the music editor app, along with a Big Hairy Audacious Goal (BHAG).
- Refinement of BHAG: The user provided a BHAG example and requested a toned-down version to fit the project context.
- Summary of Project Report Specs: The user provided specifications for the Project Report, outlining its focus, contents, purpose, and key elements.
- Addition of New Sections to Project Report: Brief additions were made to each section of the Project Report, addressing the potential execution of testing and evaluation, results and discussion, and conclusion.
- User Manual Creation: The user requested the creation of a user manual for the music editor app, adhering to the provided specifications.
- Detailed Summary Request: Finally, the user requested a detailed summary of the entire chat history, which led to the current overview.

File Name: ChatGPT_Logs_for_Sohum-2.html
Brief Summary: In our discussion, we embarked on a comprehensive journey to enhance your Java Swing music notation editor application. Here's a summary of the key points and actions taken:

- Project Overview: You described your project as a music notation editor aimed at music students and educators, focusing on creating and editing musical scores with functionalities like placing, moving, removing notes, and basic playback.
- Feature Implementation and Code Refinement: We discussed implementing drag-and-drop functionality for music symbols, ensuring symbols could be repositioned on the staff with their coordinates updated for accurate playback. Detailed code modifications were provided to add drag functionality, including changes to mouse event handlers and ensuring symbols' pitches were recalculated upon repositioning. Refactoring advice was given, emphasizing good software design practices and the application of design patterns such as the Command Pattern for actions like play, pause, save, and load. We introduced serialization for saving and loading the state of musical compositions, allowing users to preserve their work.
- Command Pattern: Used for decoupling UI actions (play, pause, save, load) from their execution logic, enhancing flexibility and extendability.
- Single Responsibility Principle and Separation of Concerns: Achieved through extracting mouse event handling into separate listener classes and defining clear roles for each class and method.
- Serialization: Employed for saving and loading compositions, encapsulating the state of musical scores in a user-friendly manner.
- Save and Load: Implemented serialization-based mechanisms to save and load compositions, including UI elements for user interaction.
- File Management Choices: Enhanced the save feature to give users the option to overwrite an existing file or save as a new one after editing a loaded composition.
- Technical Solutions and Code Examples: Provided specific code examples and modifications for implementing requested features and refactoring existing code, ensuring clarity on how to integrate these changes into your application.
- Discussion on Good Software Design: Highlighted the importance of adhering to software design principles to make the application more robust, maintainable, and user-friendly.Throughout our conversation, we focused on enhancing the functionality and architecture of your music notation editor, ensuring it not only met the initial project goals but was also structured in a way that promotes good software design practices
- User Experience Enhancements: Identified areas for improvement included visual aspects of the application, MIDI performance, and intuitive note placement, highlighting the need for further refinement to elevate user interaction.
- Conclusion and Reflection: We summarized the project's achievements, challenges, and learnings, acknowledging the successful implementation of foundational features while recognizing the potential for future enhancements to deepen the application's educational and functional capabilities.

File Name: ChatGPT_Logs_for_Emily.html
Brief Summary: This was my main chat that I used in Weeks 1, 3, and 4 of the process as well as the submission week.

- UML Diagram: Suggestions on how to create the classes and what classes should be involved as well as what the diagram should look like.
- Design Patterns: Suggestions to what design patterns can be enforced or practiced in the project.
- Mouse Interaction: Debug and add more mouse interactions as to remove the note after it has been able to be placed down. Making sure symbols are redrawn correctly.
- Drawing Enhancements: Various improvements were made to the drawing methods such as resizing images, enhancing font styles, drawing the time signature, and refining the flag of the eighth note.
- Symbol Interaction: Functionality was added to handle interactions with symbols such as cloning symbols and adjusting volume based on dynamic symbols. Figuring out how the sound system would work with adjusting of volume throughout the score rather than just for a period of time.
- Pitch Calculation: Adjustments were made to the pitch calculation logic that was implemented by Sohum (and ChatGPT) to ensure accurate mapping between Y coordinates on the staff and MIDI pitch values. Extensive testing of different functions and methods were made before resulting in a Map implementation of the pitches.
- Other Enhancements: Additional improvements were suggested, including optimizing code for readability and efficiency (practicing art of clean code), refining user interface elements, and handling edge cases like the pitch of no sound (rest notes).

File Name: ChatGPT_Logs_for_Emily-2.html
Brief Summary: This was a shorter chat (meant to start ChatGPT off on a clear mind) that I used for week 2 in attempting to receive debugging feedback of less bias since using the main chat generated poor advice.

- Initial Request: Asked for an update on my code by sharing the current code (at that time) to fix the ‘StaffPanel’ and the clicking of a music note on the toolbar and clicking the position on the panel.
- Drawing Staffs: Got information on how to improve the drawing of the staffs and make it so that the lines were correct and the spacing was optimal. Also try to keep the functions of drawing to a limited number of lines.
- Music Note clone(): Attempt to make the ability of mouse clicking into the StaffPanel to keep good placement logic and selection handling. Attempts were made as well to fix the formatting and missing code sections of the StaffPanel code.