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 management of the music score, toolbars to add measures, and musical elements like time signature and playback. The main interface will just include the music sheet that can span five measures wide and will iterate to create new rows of measures downwards. Most interactions will be determined through right and left clicks of the mouse. The architecture employs a modular approach, allowing for the seamless integration of various components. In our current implementation, we have created the score sheet in a simpler manner. The addition of new measures is a feature that is in the works and attached in a separate branch of the Git repository. Additionally, pitch calculation and playback have been implemented in its own branch. Currently, it has not been merged into the main application.

Design Patterns

Singleton: Utilized in the Playback class to ensure the existence of only one instance responsible for MIDI playback. Similarly, a singleton instance is maintained for the entire music score, controlling its state and history within the MusicNotationEditorUI class.

Factory: Employed in the creation of different musical elements, such as notes and rests, within the Toolbar class. This pattern facilitates the dynamic instantiation of various types of musical elements based on user interactions.

Observer: The Observer design pattern is used to keep the music score updated dynamically in response to changes in its underlying data. Components observing the music score, such as graphical representations or playback controls, automatically reflect changes made to the score, ensuring consistency between different views of the data. By decoupling observers from the subject (i.e., the music score), the Observer pattern promotes modularity and scalability, allowing new observers to be added without modifying existing code.

State: Utilized in the Playback class to manage the playback state, allowing transitions between playing and paused states seamlessly. This pattern enhances the flexibility of the playback functionality.

Prototype: This design pattern is simply used in the MusicSymbol class in the look of the clone() function. This function is just used to clone the notes accordingly to get on the phrase.

Command:

The Command Pattern was introduced to encapsulate playback actions (Play and Stop) as command objects. This pattern provides a clean way to handle UI actions.
Concrete command classes (PlayCommand and StopCommand) were created, and their integration with the UI buttons was outlined and will be integrated.

Components Descriptions:

MusicNotationEditorUI: The central class 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.

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.

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 clefs and signatures. It offers functionality to add additional empty phrases for composing music notation. More changes may be made to the logic behind the Phrase and Measure relationship.

Measure - Handles note management within a measure. This class allows users to place notes and rests at specific positions within the measure.

MusicalSymbols - 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.

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.

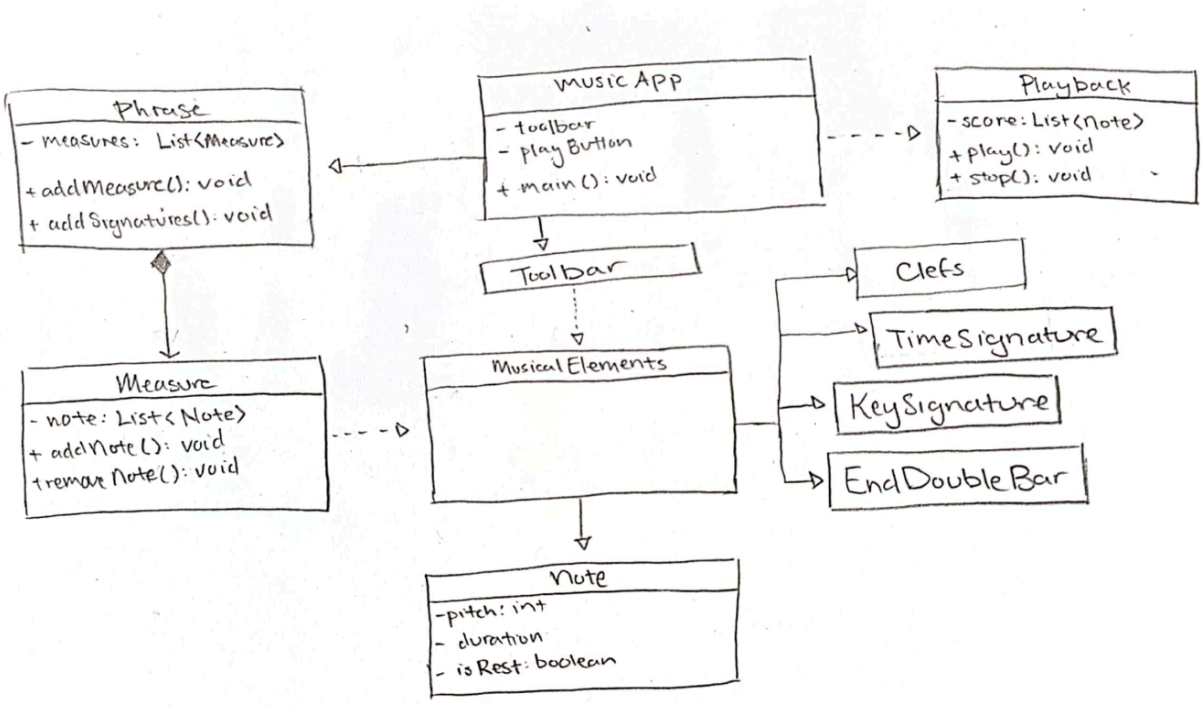
Clefs, TimeSignature, EndDoubleBar - Currently these classes are mostly for visuals but as the project progresses, complexity can be increased and additions can be made.

KeySignature - This class is mostly to draw the key signature the user decides to select. It will also alter the pitch accordingly. This will have to include updating the music score and the notes correctly. Difficult to implement but will be rewarding for music creation.

Many of these aren't implemented yet but are part of the plan. Details on the code are reflected in our Git repository. Current updates are a refinement of the interface. Our select and drop notes action is currently working. The notes' cloning and the collection of its coordinates upon clicking on the staff works. We worked on perfecting that and then creating the ability to drag symbols before moving will be implemented. For now, we have completed the playback feature.

Interactive versions of the staff, measure, and phrase are being worked on but if things don't go through it would be better to move on from that and simply display the measures without letting the user have control over it. Focus on the note being able to be played and put on the score is more important.

UML Diagram:



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.

Currently, the structure may shift and change due to this version being a work in progress. Multiple ways are being tested out to formulate a better version of the phrase and measure behaviors.

Standards and Conventions:

Descriptive Variable Names: Variable names such as playPauseButton, stopButton, 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.

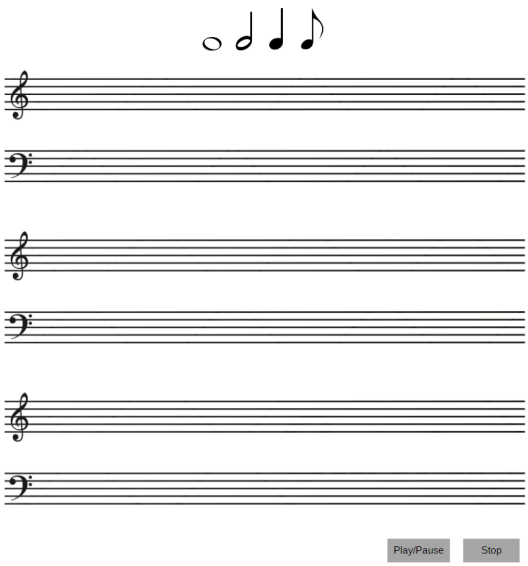
Encapsulation: Encapsulate related functionality and data, promoting modularity and reusability. For example, the StaffPanel class encapsulates the logic for drawing staff lines, while the ControlPanel class encapsulates the logic for managing control buttons.

Abstraction: Abstract away implementation details and provide a simplified interface for interacting with components. For instance, the MusicSymbol abstract class provides a common interface for different musical symbols, allowing for polymorphic behavior and code reuse.

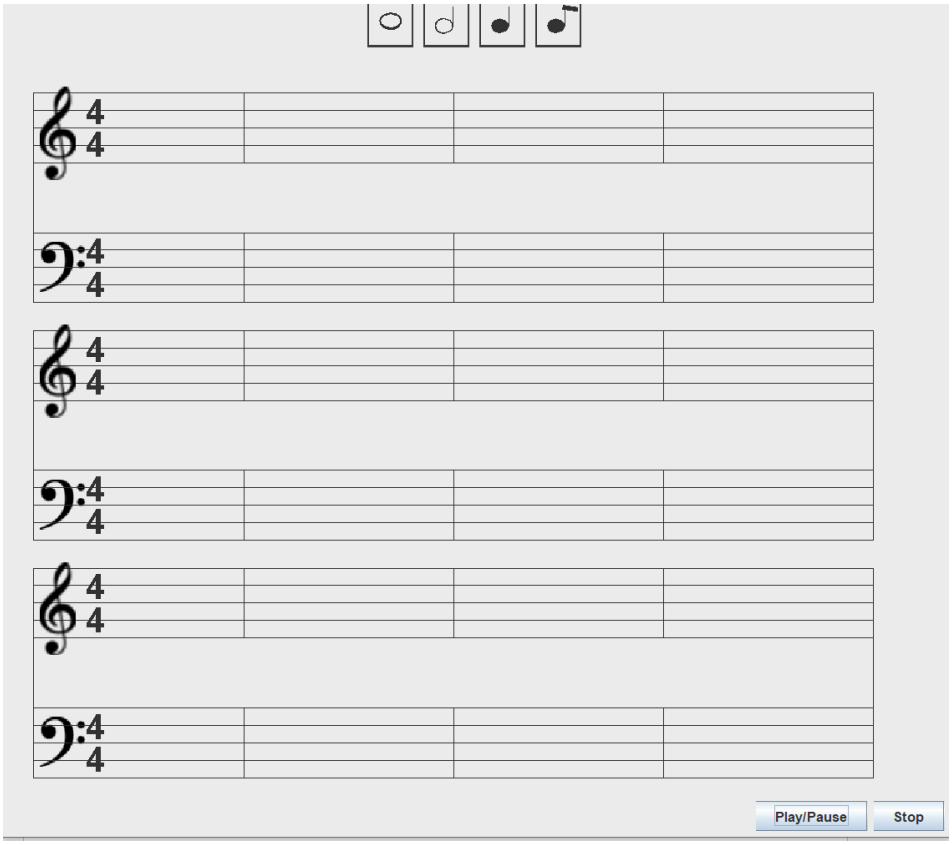
User Stories:
As a user, I should be able to see a graphical representation of a musical staff displayed on the editor interface.
As a user, I should be able to visually place notes on the staff by clicking on specific positions within the staff.
As a user, I should be able to edit existing notes by 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 a music and technology enthusiast we wish to create a comprehensive music notation editor that combines intuitive graphical interfaces with powerful functionality, enabling musicians of different levels to create, edit, and playback intricate musical scores. By leveraging innovative 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.

BASIC SKETCH OF USER INTERFACE:



Implementation:



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 and implement by using what we learned in class for Java. Key development phases align with the recommended milestone schedule:

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.

From the previous week to the latest update, we have managed to start our project and draw out some symbols on the application. We designed the staff panel and started the playback interface buttons. The notes are now selectable and we can display them on the staff as needed. Soon, we will be working on making this part of the application perfect and refactoring any code as well as organizing the class files neatly.

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:

The Command Pattern was introduced to encapsulate playback actions (Play and Stop) as command objects. For now, this is commented out as a lot of refactoring will be needed in the existing code to integrate this pattern

Testing and Evaluation:

During subsequent phases, rigorous testing strategies will be implemented to ensure the software's functionality, performance, and reliability. This will involve the creation of comprehensive test cases, including unit tests, integration tests, and user acceptance tests.

Instructions were given for adding debug print statements to help test the pitch calculation and symbol addition features.

Evaluation will focus on identifying and addressing any issues or bugs discovered during testing, as well as assessing the overall usability and effectiveness of the software.

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 musical scores.

Table of Contents:

1. Installation Instructions:

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

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

2. Overview of Features:

- Staff Display: View and edit onto 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.

3. How to Create and play a Music Score:

- Use the toolbar to add musical notes and symbols to the staff panel.
- To add a note, select the desired note duration from the toolbar and click it onto the staff.
- To move a note, click and drag it to the desired position on the staff.
- Repeat the previous step to create a sequence of various notes.
- To delete a note/symbol, simply right-click on it.
- Click on the "Play/Pause" button to start or pause the playback of your composition.
- Use the "Stop" button to stop playback and return to the beginning of the composition.

4. Troubleshooting:

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

- Ensure that your computer meets the minimum system requirements for the software.
- Check for updates to the software and install any available updates.
- If the issue persists, contact our customer support team for assistance.

5. Frequently Asked Questions (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 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.

Appendix

Files in order:
ChatGPT Logs for Sohum - https://github.com/SohumGoel/MusicEditor160/tree/SohumNew
ChatGPT Logs for Emily - https://github.com/SohumGoel/MusicEditor160/tree/EmilyRefactor

Branch to run to view playback feature -
https://github.com/SohumGoel/MusicEditor160/tree/SohumNew