EECS2311: SOFTWARE DEVELOPMENT PROJECT

Design Document

April 11, 2022

PREPARED FOR

Vassilios Tzerpos,
Students of Lassonde School of Engineering
&
Musicians

PREPARED BY

Hiba Jaleel - 215735020
Kuimou Yi - 216704819
Kamsi Idimogu - 216880288
Maaz Siddiqui - 216402927

Table of Contents

Introduction

This document is an overview of the implementation of the TAB2MXL application. We will discuss important classes and methods and how they interact with each other.

TAB2MXL is a JavaFX application with a graphic user interface that allows end users to convert text-based tablature into machine-friendly MusicXML files. This application will also preview the MusicXML as sheet music and play it as music. Moreover, it provides users the ability to customize the sheet music to their liking.

For more detail on the class and methods, you may visit the Important Class and methods section.

Overview of the System's Structure

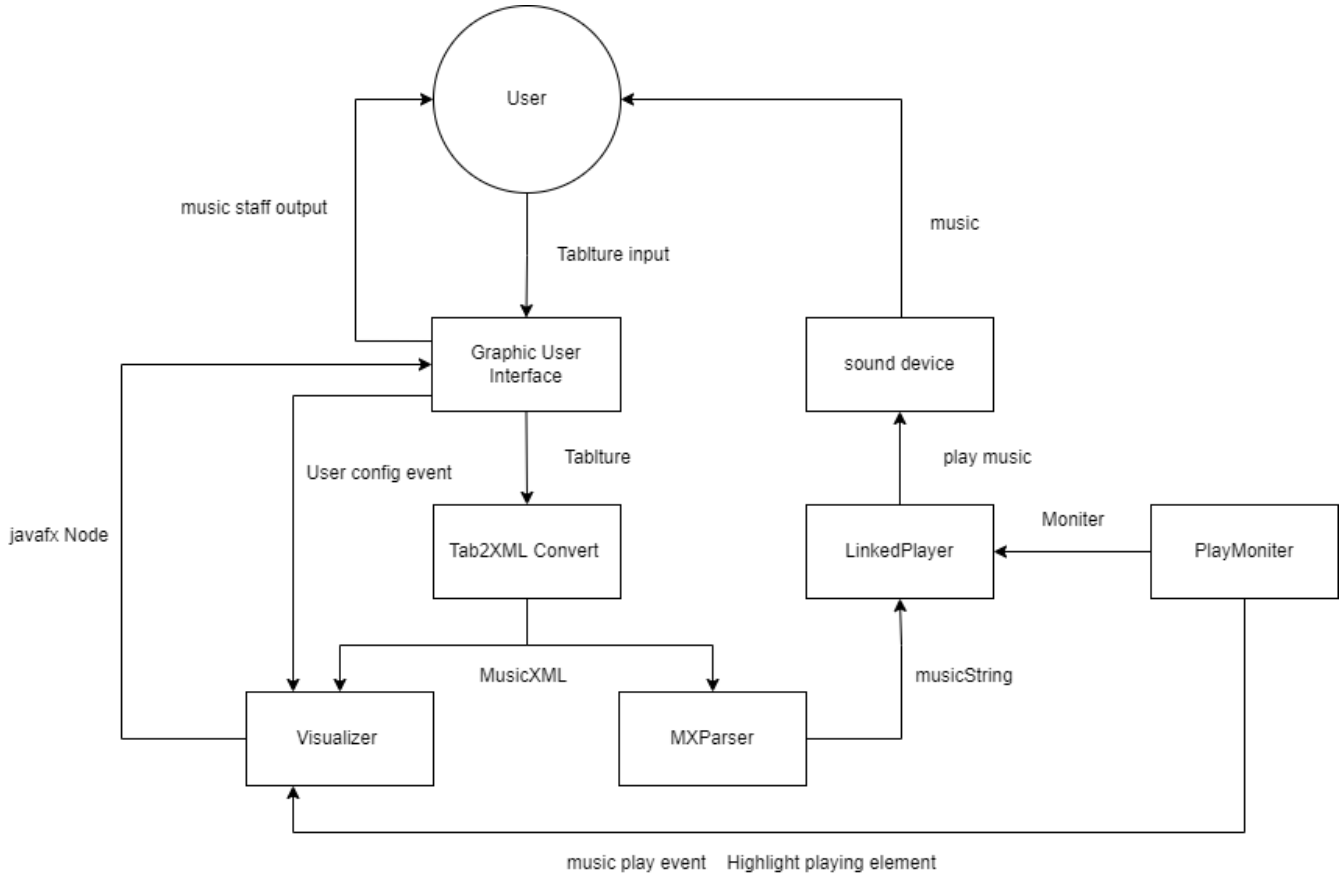


Figure 1: Overview of the TAB2XML Structure

Graphical User Interface

The graphical user interface builds on the JavaFX library with a model-view-controller pattern. The user interface uses an FXML loader to load the FXML file and link it with the controller.

Tablature to MusicXML Converter

The text-based tablature to MusicXML converter is derived from the starter code. Please visit the origin repository and view the design document if you want more information regarding the converter.

Visualizer

The visualizer will translate the MusicXML file into JavaFX nodes. The visualizer will also apply highlight events when the music is playing and alignment events when the user is aligning the sheet music.

XMLParser

The parser will translate the MusicXML file into a music String. The music String will be passed into the linkedPlayer and played.

LinkedPlayer

The linked player will play music with the JFugue player. The linkedPlayer will also create a PlayMonitor to trigger visual events when music events are happening

For more information regarding the JFugue player and music String, please visit JFugue.org.

PlayMonitor

PlayMonitor will be run synchronously with the JFugue player in a different thread. It will trigger a visual event when the timing of a given music event is about to be reached.

Libraries

This project uses the following libraries:

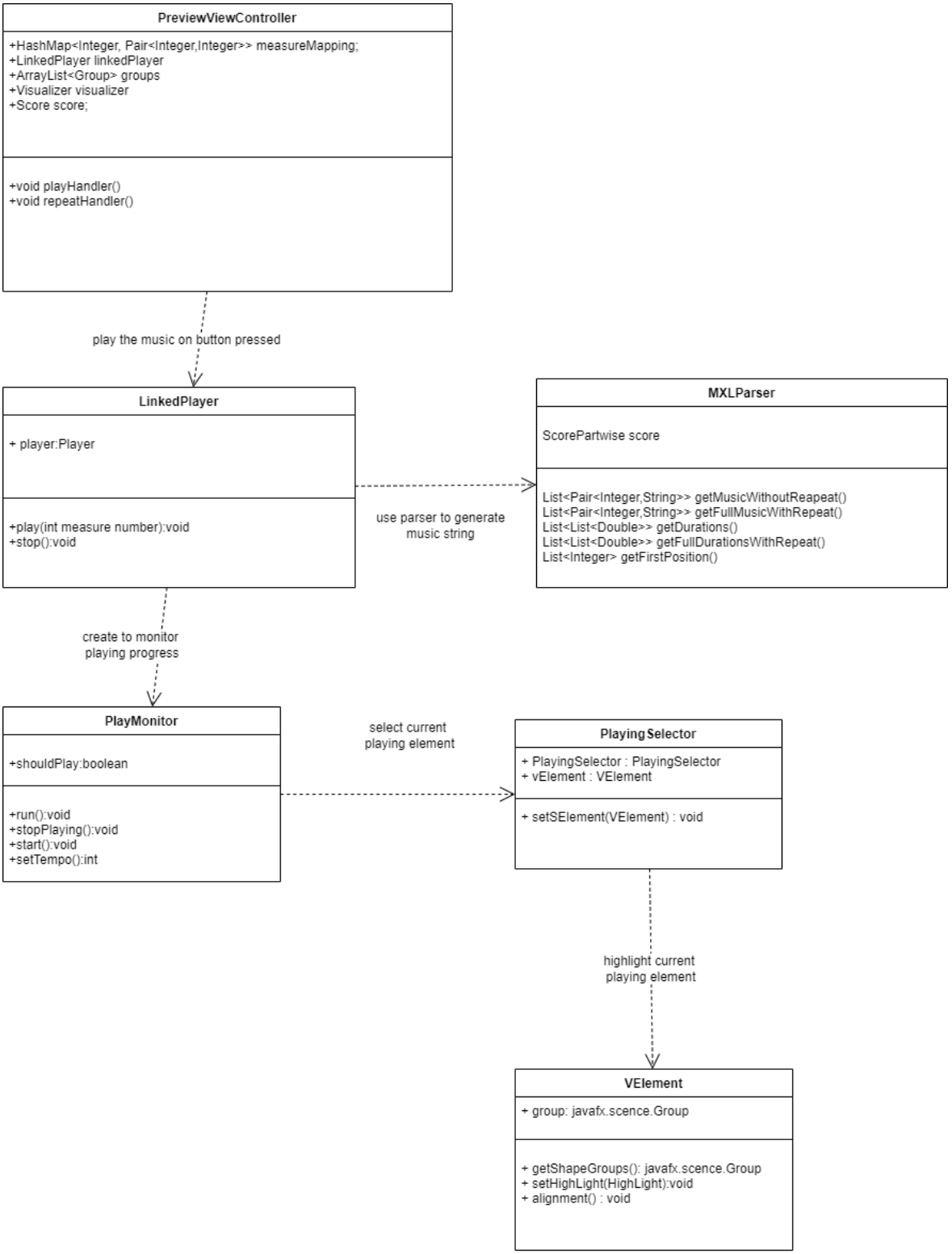
Jfugue: used to play the music
Gson: used in JSON process
JavaFX: used in the GUI
pdfBox: used in creating and exporting the .pdf file.

Design Diagrams

Class Diagrams

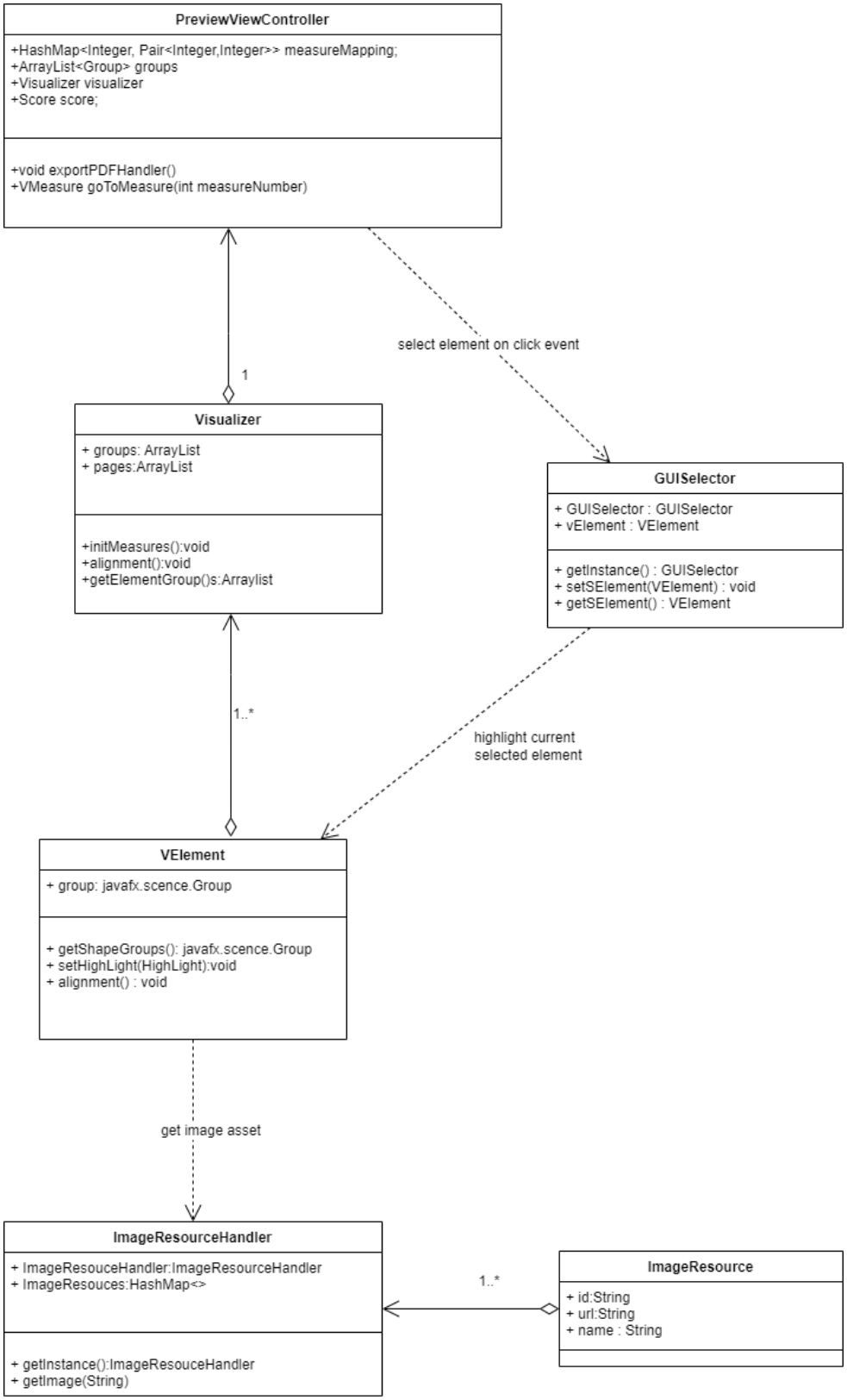
Playing Function Diagram

The full-size Class Diagram can be obtained from the link above.



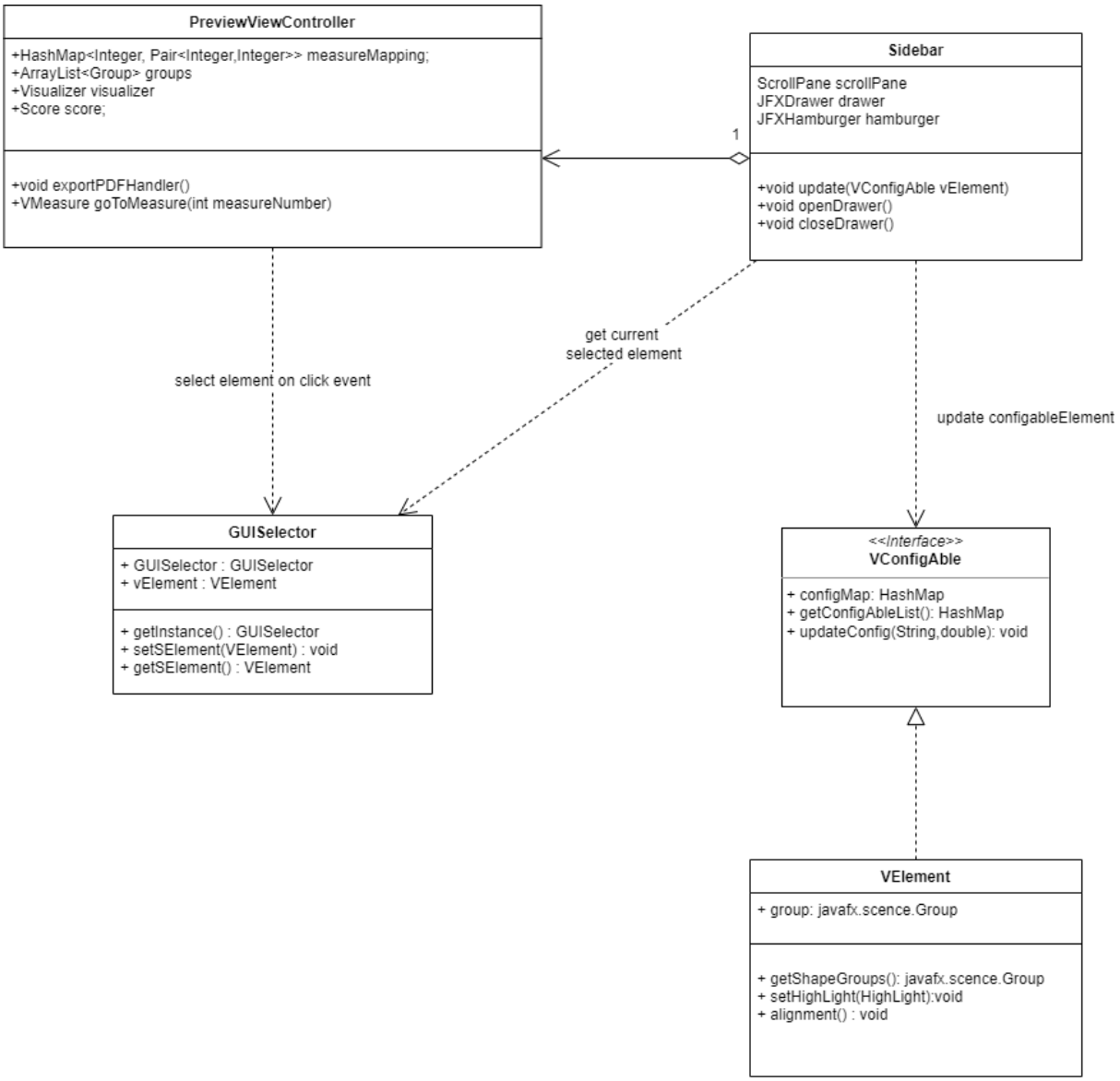
Visualization Function Diagram

The full-size Class Diagram can be obtained from the link above.



Config Function Diagram

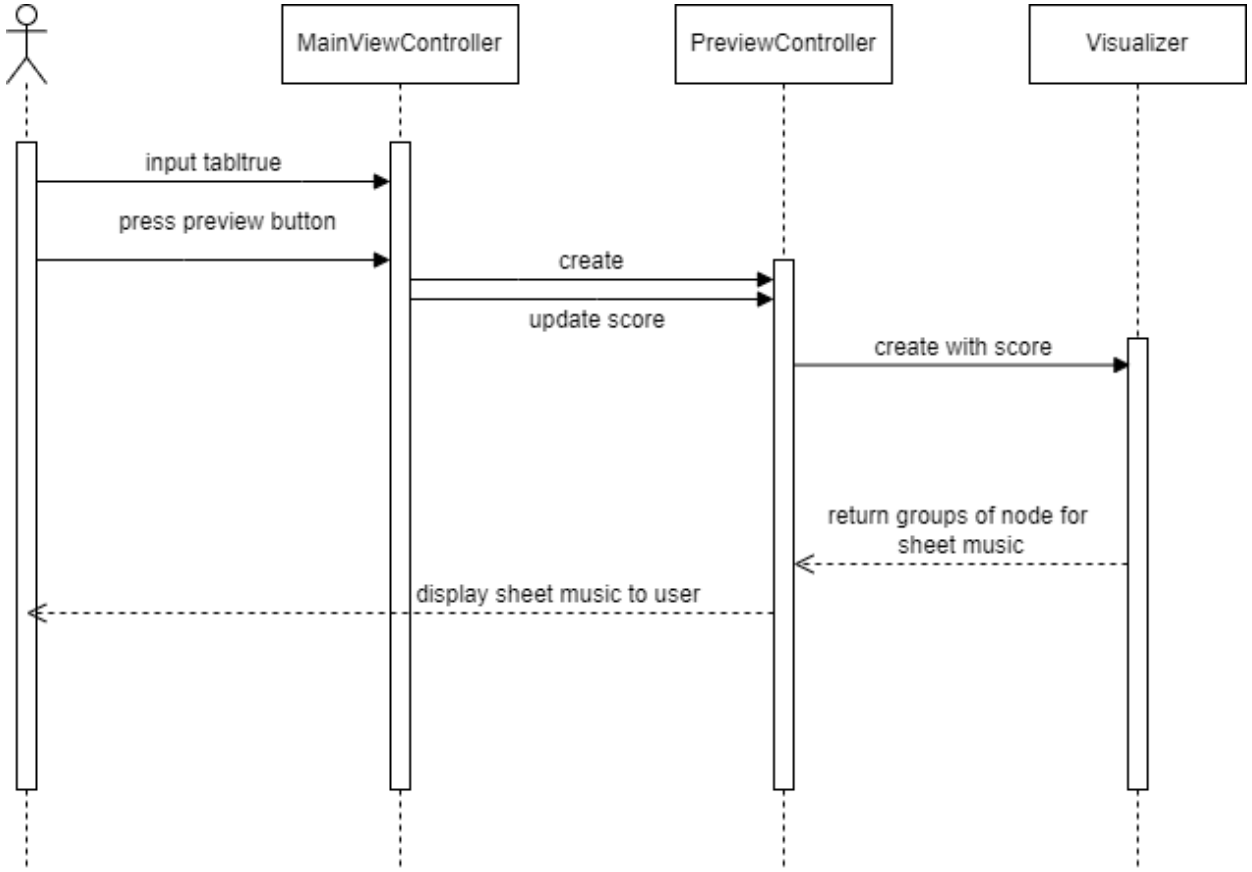
The full-size Class Diagram can be obtained from the link above.



Sequence Diagrams

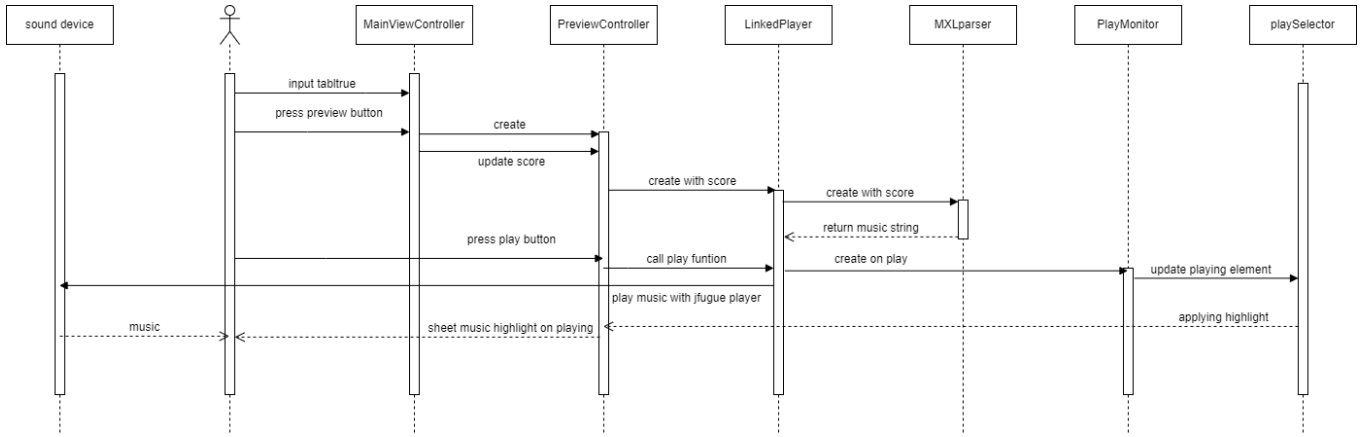
View Sheet Music

The full-size Sequence Diagram can be obtained from the link above.



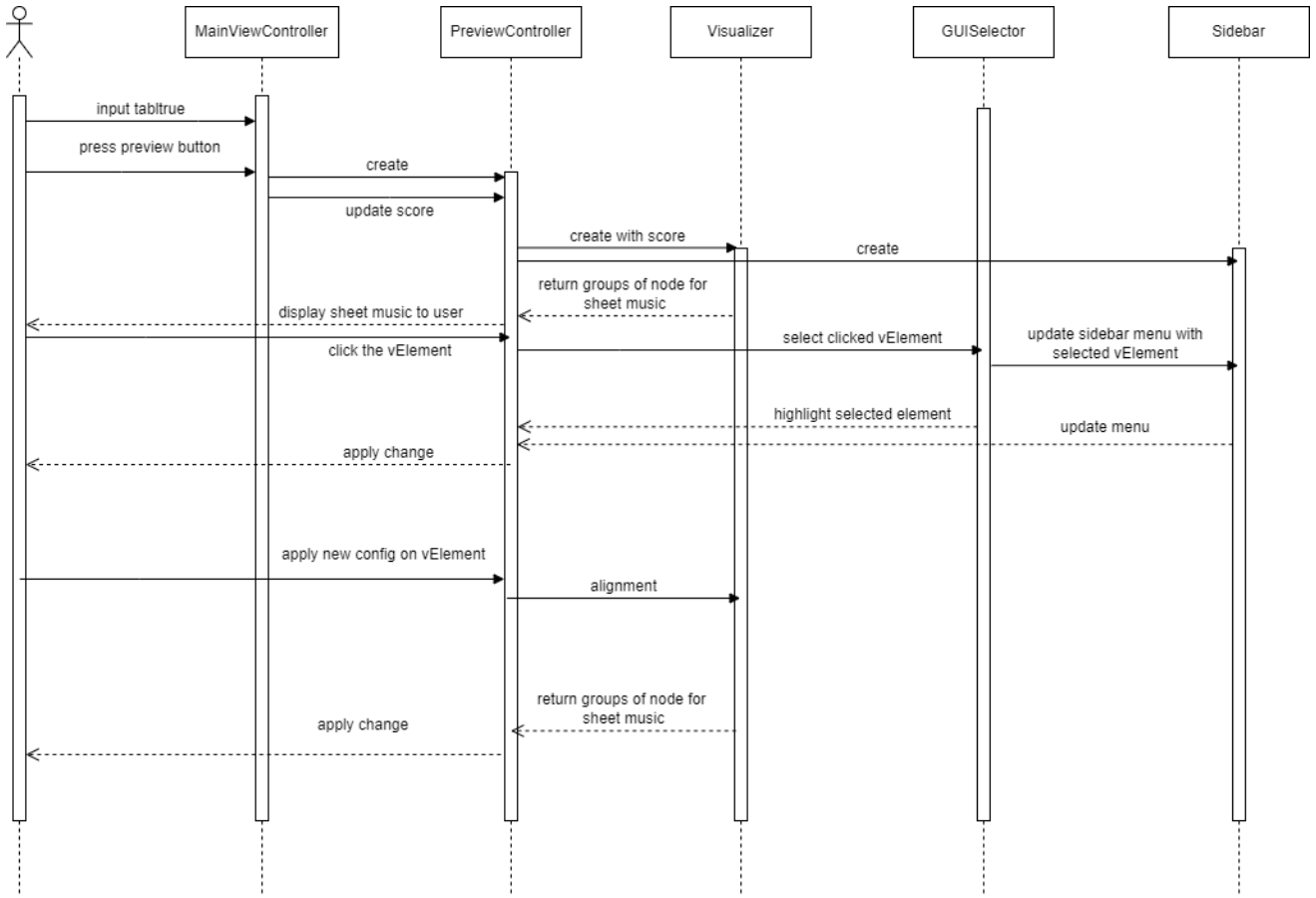
Play Sheet Music

The full-size Sequence Diagram can be obtained from the link above.



Select/Config Sheet Music Element

The full-size Sequence Diagram can be obtained from the link above.



Important Classes & Methods

GUI Functions

PreviewViewController

```
class PreviewViewController
```

PreviewViewController is the main controller for the music playing and music sheet visualizing functions.

PreviewViewController is created by FXMLoader from the MainViewController via "previewMXL.fxml."

After PreviewViewController is created, MainViewController will update its Score from the converter in the MainViewController and register an event that updates PreviewViewController in the Textarea change.

A scrollpane is implemented as the main viewport of the sheet music.

In this controller, the following functions are implemented:

- Register event and config display property for the GUI elements: spinner, button, toggle button, and drawer. The listener is applied in the spinner to execute the corresponding function on change.
- Initialize visualizer and linkedplayer with the current score. Get visual elements in the format of groups from the visualizer and set it as scrollpane's content.
- groups is a collection of Group objects and each entry represents a single page.

The following methods and their descriptions show how some of the important user action is handled by the controller:

```
void exportPDFHandler()
```

exportPdfHandler() function will be called when the export button is pressed. The following action will be executed in order to export pdf file:

1. Create a FileChooser for a user to assign a location to save the file.
2. Take a snapshot of each page for the current sheet music by group.snapshot() function that is provided by JavaFX.
3. Attach each image into a single page of a new PDFDocument via PDFbox and save this document into the desired location.

```
void setRepeat()
```

setRepeat() function will be called when the repeatButton is selected or unselected by the user. It will enable/disable repeat functionality by changing a config option in the VConfig object.

```
void playHandler()
```

playHandler() function will be called when the playButton is selected or unselected by the user. It will start or stop the music by calling linkedPlayer.play() or linkedPlay.stop().

When the program tries to play, it will check the GUISelector to see if the user has selected any note/measure. If the user selected a note/measure, the player will start playing from that measure or the measure that the selected note belongs to. Otherwise, the player will start from the beginning.

```
VMeasure goToMeasure(int measureNumber)
```

gotoMeasure function will be called when the measure spinner is changing. To set the scrollpane's viewport into the given measure, the following action is taken:

1. Acquire a measureMapping from the visualizer. It will be a Map with measure number as keys and Pair<Integer, Integer> as values. The Pair object contains the page where the current measure is located as key and which line that current measure is located as value.
2. Go to the page by assigning the corresponding group as scrollpane's content.
3. Set scrollpane's vvalue and hvalue to locate desired measure.
4. Highlight the desired measure by selecting it in the GUISelector.

```
void reset()
```

Reset function will be called when the reset button is pressed or the previewViewControll is initializing. It will create a new visualizer and new linkedPlayer to reset configs to the default.

Sidebar

class Sidebar

Sidebar is a new kind of GUI component that we created to deliver the ability to config visual elements. When a VElement is selected in GUISelector, the Sidebar.update() will be called in order to create a new config menu that is suitable for the current VElement.

The sidebar can be closed or open by the following methods:

```
void openDrawer()
void closeDrawer()
```

void update(VConfigAble vElement)

To update the config menu, the following action is taken:

1. Get config maps, including current value, limit, step, and configurable.
2. For each key in the configMap, if it is configurable, create a spinner with the current value, limit and step for that key.
3. Add a label to that spinner. Label text will be determined by VUitily.getDisplayName(String Key).
4. Add a listener to the spinner. when the value is updated, call vElement.updateConfig(String key, Double new Value).

GUISelector

class GUISelector

GUISelector is a singleton class that responds to the storage VElement that the user selected from the GUI. Also, GUISelector maintains that only one VElement can be selected at once.

```
void setSElement(VElement sElement)
```
There is a local variable VElement which stores the current selected element and it is nullable.

setSElement() will be called when the user clicks a VElement. If the new Element that the user tries to select is the same element as the local VElement, it will be deselected (local VElement set to null). Otherwise, the local VElement will be set to the new Element.

Visualization Functions

Visualizer

```
class Visualizer
```
Visualizer is the main class that creates the visual elements, stores them, as well as aligns them.

Also, the visualizer is responsible for providing information about the position of the visual elements.

```
void initMeasures()
```
initMeasures will create the VMeasures objects and its sub-nodes references to the Music XML object (Score).

Once the VMeasure objects and their sub-nodes are created, they will continue to exist until the user resets everything to default or change tabs.

This approach will allow VMeasure and its sub-objects to store information about their alignment configuration and for the sub-ojbects to not lose the information during realignment.

```
void alignment()
```

alignment() will be called after the user changes the configuration of some element.

VElement.alignment() will be called in order to apply the change, (VMeasure and its sub-nodes are VElement).

After the change is applied, the program will try to fit the measures in the new lines and pages, while also creating a new mapping of the measures regarding their position in the page/line.

The algorithm that checks whether measures fit measures can be described as the following pseudocode:

```
if(Line.canFit(Measure)){
                line.fit(measure)
}else{
        if(Page.canFit(line)){
                page.fit(line);
                line = new Line;
                line.fit(measure);
}else{
        page = new Page
        page.fit(line);
                line = new Line;
                line.fit(measure)
}
}
```

VConfigAble

```
public interface VConfigAble
```

This class is an interface. Objects which the user can adjust will be implemented through this interface. Each VconfigAble object has several internal maps to store the following settings:

- the current value
- the lower limit
- the upper limit

- if it is configurable
- amount to increment the spinner by
- amount to decrement the spinner by

These are the abstract methods in this class:

```
abstract HashMap<String,Double> getConfigAbleList( );
```

ConfigAbleList stores the option as a key and its current value as the value.

```
abstract HashMap<String, Pair<Double,Double>> getLimits();
```

Limits stores the option as a key and its lower/upper limit pair as the value.

```
abstract HashMap<String,Boolean> getConfigAble();
```

ConfigAble stores the option as a key and if this option can be modified by the user as the value.

```
abstract HashMap<String,Double> getStepMap();
```

StepMap stores the option as a key and the amount to increment or decrement by, per step as value.

```
abstract void updateConfig(String id,double value);
```

Updates the current value for given the key to the new value.

VElement

```
public class VElement implements VConfigAble
```

VElement is the superclass of all important visual elements.

The following classes extend from vElement, and they have a tree relationship described as follow:

➢VPage
    ➢Vline
        ➢ VMeasure
            ➢VSign(VTime)
            ➢VNote

➢VNoteHead
➢VDot
➢VBarline
➢VCurvedNotation(use for both tied and slur)
➢VGNotation(VDrumGnotation&VGuitarGNotation)
➢VSign(VClef)

Each VElement stores its graphic elements in Javafx.group note.
and contains its child element's group.

```
public void setHighLight(HighLight states)
```

Highlights this VElement and its child element. This implementation is different from VElement to VElement

Color of highlight is determined by enum class HighLight
There are three type of states:

- PLAY
- SELECTED
- NULL

VConfig stores color for each state.

```
public Group getShapeGroups()
```

Returns the group note that contains all graphic elements.

```
public void alignment()
```

Alignment for this VElement: puts all graphic elements into correct position reference to it's Config.

ImageResourceHandler

```
public class ImageResourceHandler
```

ImageResourceHandler is a singleton class that helps VElement read and use Image Assets. Image Asset is stored as an ImageResource object in a JSON file.

ImageResource objects contain this Asset's ID and location.

When creating ImageResourceHandler, it loads ImageResource objects from a JSON file called imageList.json. After getting the Asset's location, it loads an image and stores it on a map, where the Asset's id is the key and the Image object is the value.

Music Playing function

MXLParser

```
public class MXLParser
```

This class is responsible for parsing through musicXMLs, storing them as Score objects, then into a music String.

The resulting music String will be stored measure by measure in a list to support playing from the measure feature.

A mapping called FirstPosition will record the measure number as key and its start position in the list as the value.

The resulting music String has two versions:

- the version without repeat (repeat is disabled)
- the version with repeat (repeat is enable)

There is another List that stores the duration value for each note. The play monitor will use this list to synchronize with the JFugue player in order to trigger the visual events at the right timing.

LinkedPlayer

```
public class LinkedPlayer
```

This class is responsible for playing music with music String.

When creating LinkedPlayer, XMLParser will be called to generate music strings.

Play music
When the GUI calls the play (measure number) function, the LinkedPlayer will first determine if the repeat is enabled from VConfig.
If repeat is enabled, LinkedPlayer will construct a pattern from a List of music strings with repeat.
Otherwise, it will construct a pattern from a List of music strings without the repeat.

The pattern is constructed from a specific position to the end of the list.
This specific position is determined by FirstPosition mapping from XMLParser.

After the pattern is finished, it will be assigned a tempo from VConfig.
A new PlayMonitor will be created with the duration list and tempo.
Same with the music string, the type of duration list is determined by whether repeat is enabled.

Now the music is ready to play.
PlayMonitor and Player will start simultaneously.

Stop music
To stop music, void stop() will be called.
This method will call stop() method in the JFugue player and call stopPlaying() methods in the PlayMonitor.

PlayMonitor

```
public class PlayMonitor extends Thread
```

Play Monitor is a monitor thread that synchronizes between the player and the visual element.

When the first note is playing, it will highlight it, sleep for a given duration, and highlight the next note.

The duration of the note is obtained from the list of duration that is passed by the LinkedPlayer.

PlayingSelector

```
public class PlayingSelector
```

PlayingSelector is similar to GUISelector. but it will highlight elements with HighLight.Play. and PlayingSelector will not update the sidebar menu.

Utility Functions

VUtility

```
class VUtility
```

This class contains help methods that help the visualizer determine the position, the type and the resource.

```
public static int getRelative(String step,int octave)
```

This method gets the relative position of the given step and octave, relative to E5 (which is the top line of the measure). For example, F5-E5 = -1.

```
public static String getDrumAssetName(Note note)
```

This method gets the Asset name from a given note. The Asset name is a String that the program can get from an image from the ImageResourceHandler.

```
public static int NoteType2Integer(String type)
```

This method will convert notes into String into an integer duration value. It will only return the denominator.

```
public static String getDisplayName(String id)
```
This method will convert the internal String id into a display name.

Vconfig

```
class VConfig implements VConfigAble
```
This class is a singleton class that implements VConfigAble and stores global configuration for the program.

In addition to the Double value stored by VConfigAble, it also stores the following settings:

- Color for HighLight states:
    - PLAY,
    - SELECTED,
    - NULL.
- Current instrument
- Current StaffLine Detail
- Background Color
- Whether repeat is enabled

Maintenance Scenarios

Translating/Localization

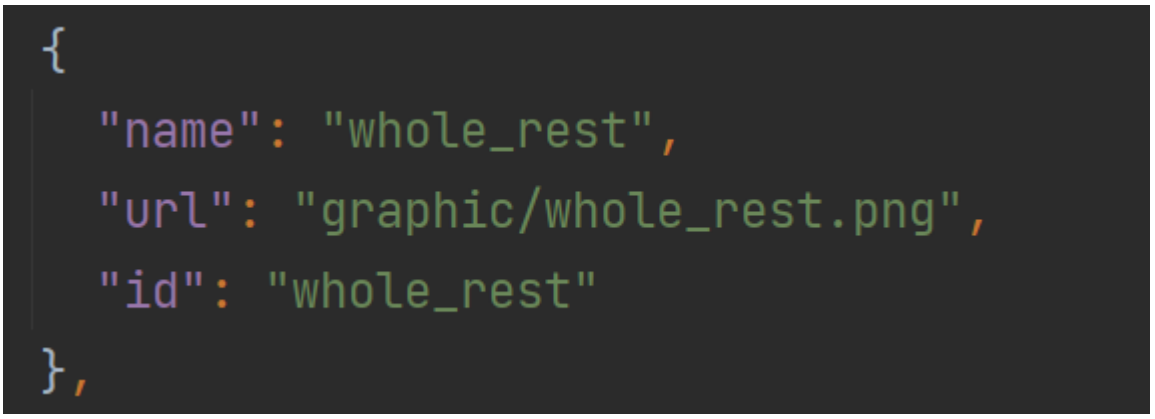
Our project provides an easy way to translate displayed texts.

1. You have to store the language type that the system is currently using in the configuration, which is the global setting.
2. Go to VUtility.getDisplayName() and return the translated text with your current language setting.
3. Notice that the label and text from the FXML loader will not go through this way. So you will have to set it manually.

Edit/Add Image Asset

Our project provides an easy way to change/edit image assets.

1. To edit an image asset, find the image asset that you want to change in the imageList.json. For example, I want to change the file named "whole_rest."



2. Change its URL to your new image asset.

   Note: Your image asset should be saved in the graphic folder in the format of a .png file. Otherwise, we cannot guarantee that the image asset will be read correctly.
3. To add an image asset, append the following text in the imagelist.json:

```
{
   "name": "$(asset's name for human to read)",
   "url": "graphic/$(asset name)",
   "id": "$(asset's id for program to get it in the ImageResourceHandler)"
  },
```

Adding a new VElement

If you want to add more notation in the sheet music, no problem!

1. Create your object that extends from VElement.
2. Initialize the graphic element that you need for this VElement in the constructor.
3. Override the alignment and add code to align the graphic element.
4. Add this VElement into a suitable parent node in its constructor.
5. Add this VElement.getShapeGroup to the parent node's group.
6. Call this Element. alignment in the parent node's alignment function.

   You have added a new VElement!