EECS2311: SOFTWARE DEVELOPMENT PROJECT

Design Document

March 30, 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

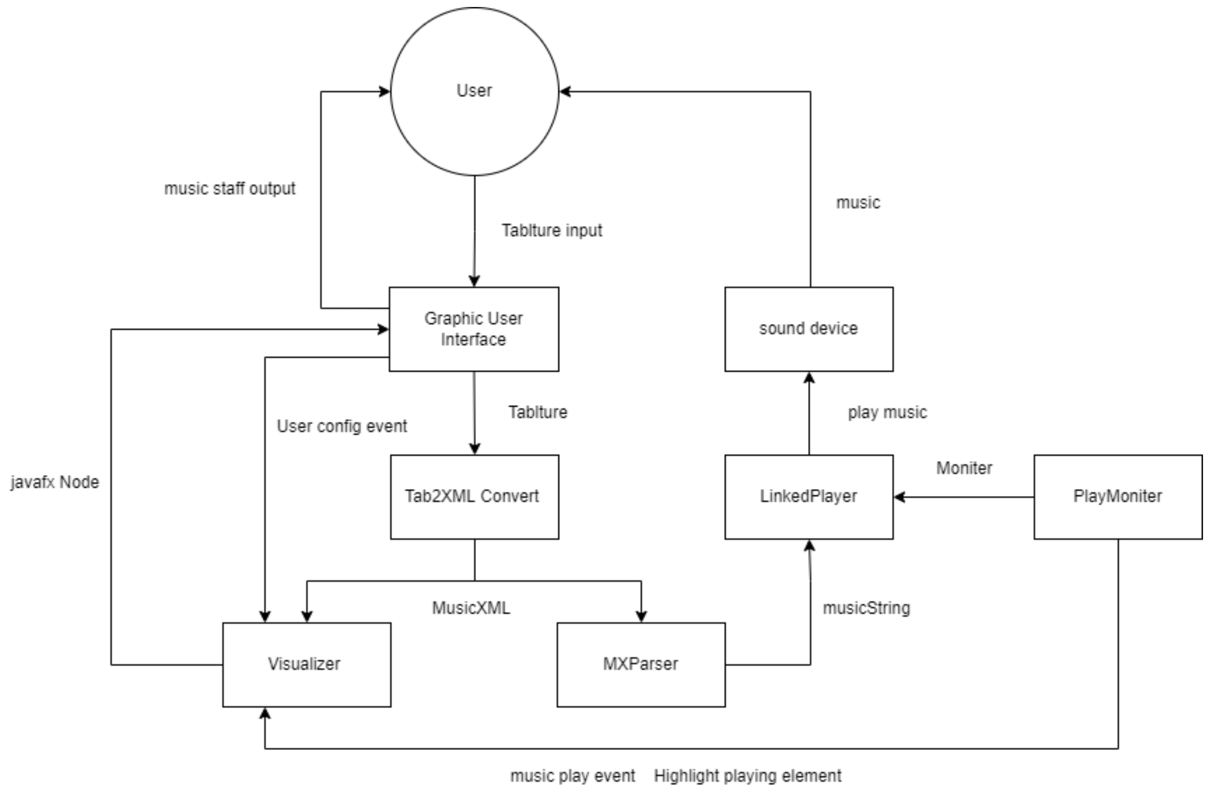
Introduction

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



Graphical User Interface

The graphical user interface builds on the JavaFX library with a model-view-controller pattern. The user interfaces use an FXML loader to load the FXML file and link them 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. Also, the visualizer will apply highlight events when the music is playing and alignment events when the user is aligning the sheet music.

MXLParser

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 JFugue player. also, the linkedPlayer will 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 reaching.

library

This project uses the following libraries:

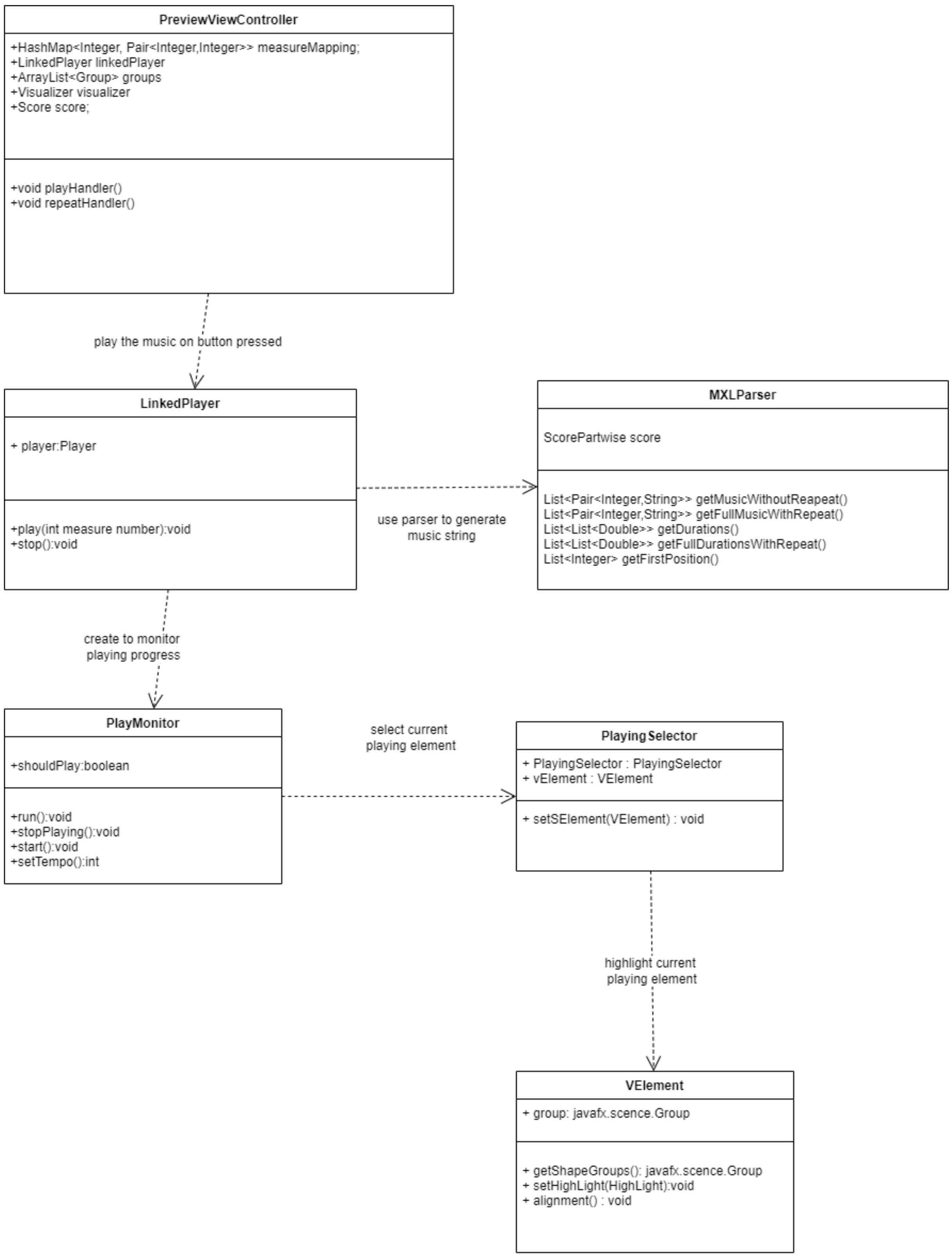
Jfugue: used in music playing
Gson: used in JSON process
JavaFX: used in GUI
pdfBox: used in creating and exporting pdf file.

Design Diagrams

Class Diagram

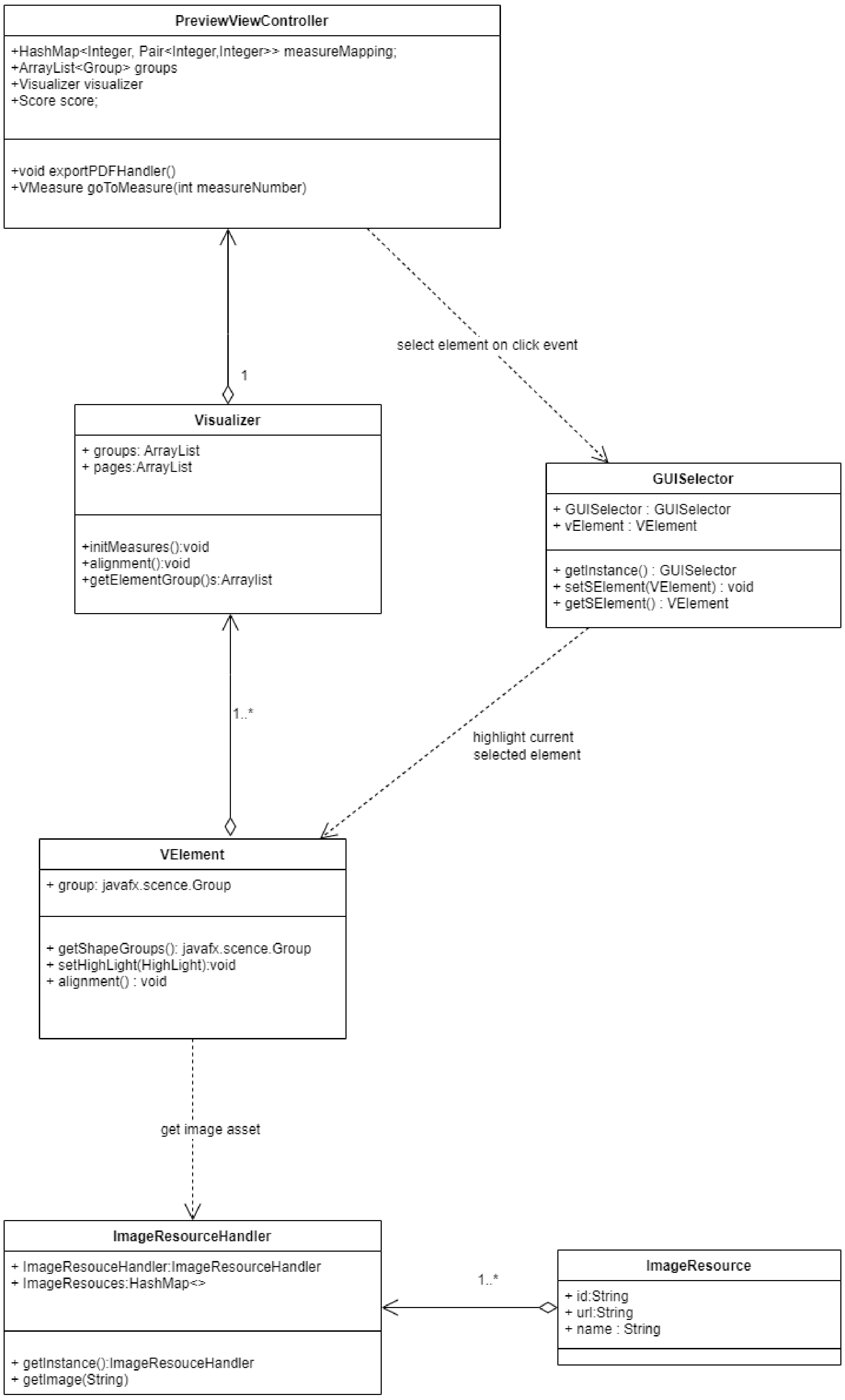
Playing Function Diagram

Full-size Class Diagram can be obtained from the link above:


PreviewViewController
+HashMap<Integer, Pair<Integer,Integer>> measureMapping;
+LinkedPlayer linkedPlayer
+ArrayList<Group> groups
+Visualizer visualizer
+Score score;
+void playHandler()
+void repeatHandler()
play the music on button pressed
LinkedPlayer
+ player:Player
+play(int measure number):void
+stop():void
use parser to generate music string
MXLParser
ScorePartwise score
List<Pair<Integer,String>> getMusicWithoutReapeat()
List<Pair<Integer,String>> getFullMusicWithRepeat()
List<List<Double>> getDurations()
List<List<Double>> getFullDurationsWithRepeat()
List<Integer> getFirstPosition()
create to monitor playing progress
PlayMonitor
+shouldPlay:boolean
+run():void
+stopPlaying():void
+start():void
+setTempo():int
select current playing element
PlayingSelector
+ PlayingSelector : PlayingSelector
+ vElement : VElement
+ setSElement(VElement) : void
highlight current playing element
VElement
+ group: javafx.scence.Group
+ getShapeGroups(): javafx.scence.Group
+ setHighLight(HighLight):void
+ alignment() : void

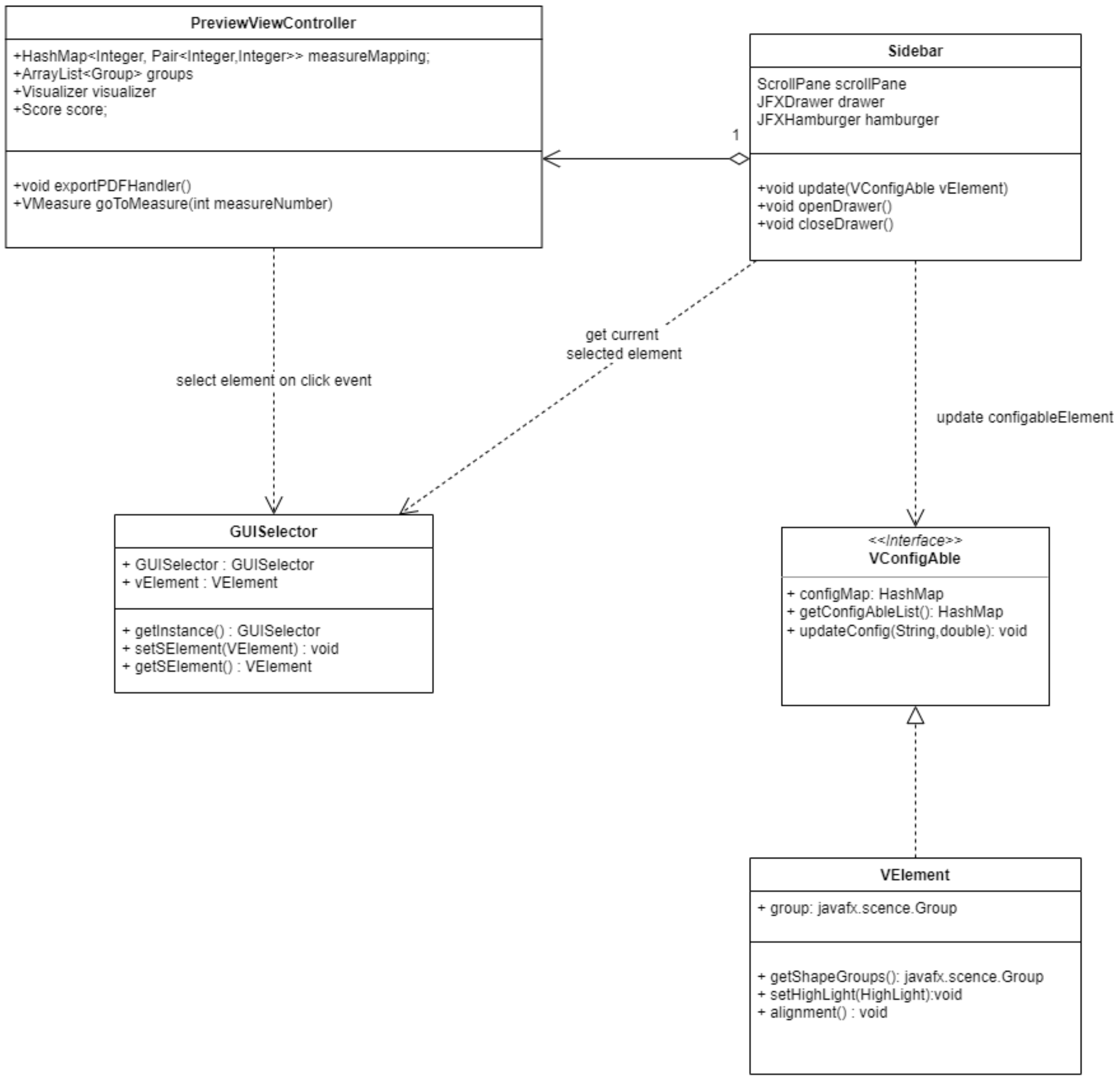
Visualization Function Diagram

Full-size Class Diagram can be obtained from the link above:


PreviewViewController
+HashMap<Integer, Pair<Integer,Integer>> measureMapping;
+ArrayList<Group> groups
+Visualizer visualizer
+Score score;
+void exportPDFHandler()
+VMeasure goToMeasure(int measureNumber)
select element on click event
1
Visualizer
+ groups: ArrayList
+ pages:ArrayList
+initMeasures():void
+alignment():void
+getElementGroup()s:Arraylist
GUISelector
+ GUISelector : GUISelector
+ vElement : VElement
+ getInstance() : GUISelector
+ setSElement(VElement) : void
+ getSElement() : VElement
1..*
highlight current
selected element
VElement
+ group: javafx.scence.Group
+ getShapeGroups(): javafx.scence.Group
+ setHighLight(HighLight):void
+ alignment() : void
get image asset
ImageResourceHandler
+ ImageResouceHandler:ImageResourceHandler
+ ImageResouces:HashMap<>
+ getInstance():ImageResouceHandler
+ getImage(String)
1..*
ImageResource
+ id:String
+ url:String
+ name : String

Config Function Diagram

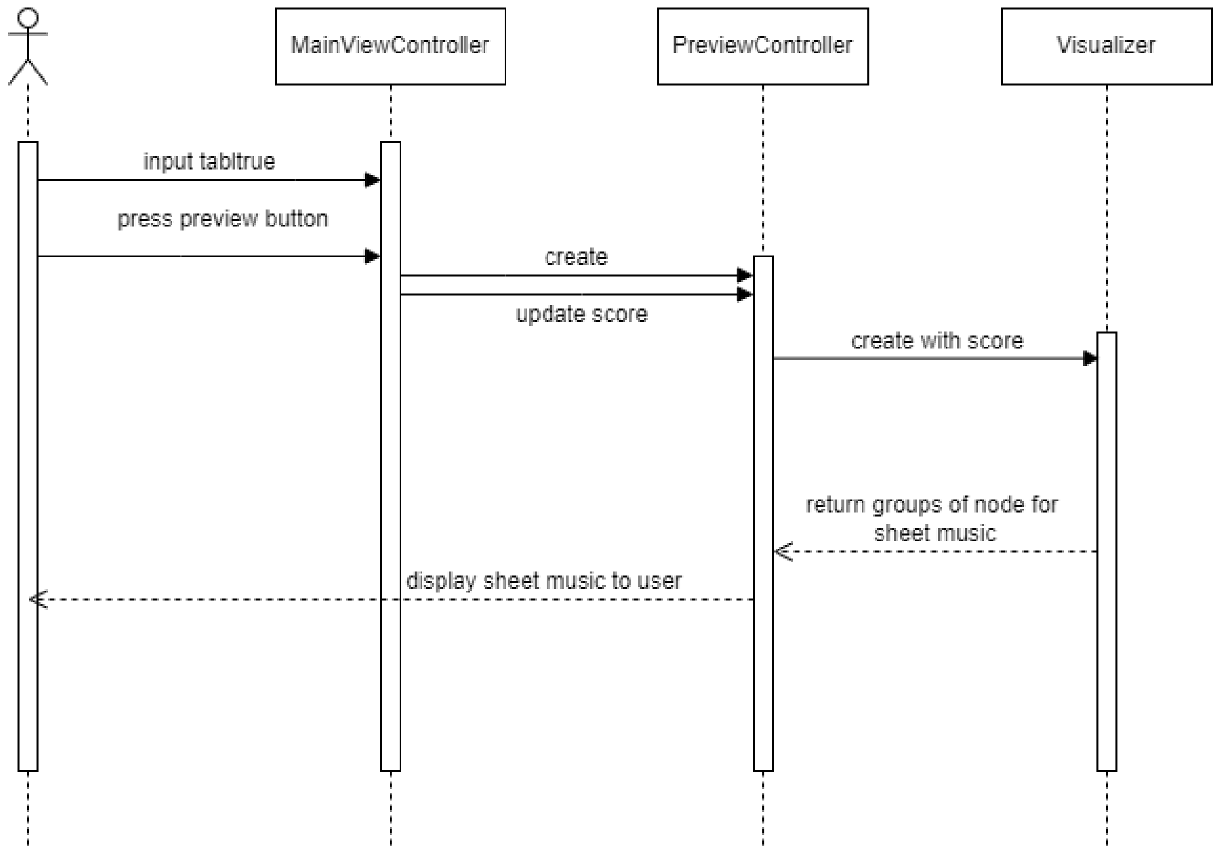
Full-size Class Diagram can be obtained from the link above:


PreviewViewController
+HashMap<Integer, Pair<Integer,Integer>> measureMapping;
+ArrayList<Group> groups
+Visualizer visualizer
+Score score;
+void exportPDFHandler()
+VMeasure goToMeasure(int measureNumber)
Sidebar
ScrollPane scrollPane
JFXDrawer drawer
JFXHamburger hamburger
1
+void update(VConfigAble vElement)
+void openDrawer()
+void closeDrawer()
get current selected element
select element on click event
update configableElement
GUISelector
+ GUISelector : GUISelector
+ vElement : VElement
+ getInstance() : GUISelector
+ setSElement(VElement) : void
+ getSElement() : VElement
<<Interface>>
VConfigAble
+ configMap: HashMap
+ getConfigAbleList(): HashMap
+ updateConfig(String,double): void
VElement
+ group: javafx.scence.Group
+ getShapeGroups(): javafx.scence.Group
+ setHighLight(HighLight):void
+ alignment() : void

Sequence Diagram

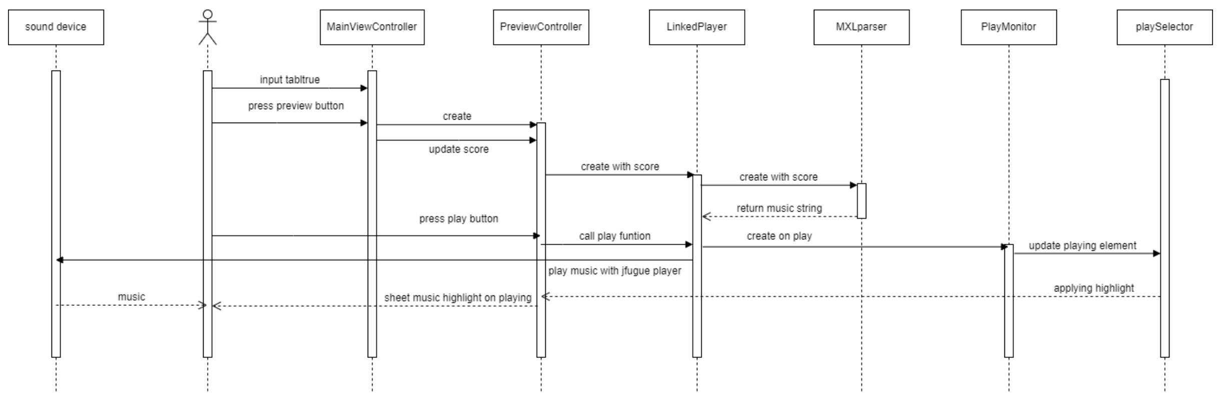
view sheet music

Full-size Sequence Diagram can be obtained from the link above:



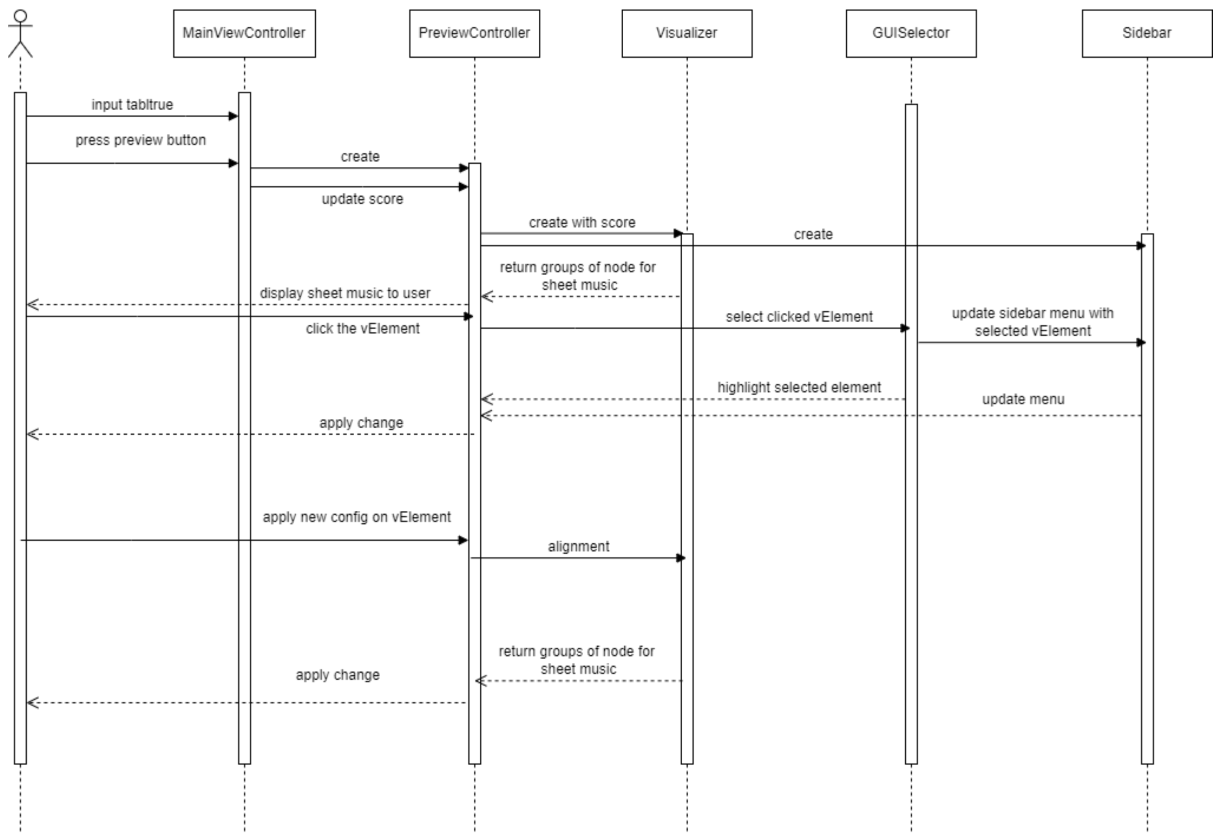
play sheet music

Full-size Sequence Diagram can be obtained from the link above:



select/config sheet music element

Full-size Sequence Diagram can be obtained from the link above:



Important Class and Methods

GUI functions

PreviewViewController

---

class PreviewViewController

PreviewViewController is the main controller for playing and 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 on Textarea change.

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

in this controller, the following important function is 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.

Following methods and their description shows how some of the important user action is handled by 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 saving location
2. take a snapshot of each page for current sheet music by group.snapshot() function that provides by JavaFX.
3. attach each image into a single page of a new PDFDocument via PDFbox and save this document to desired location

```
void setRepeat()
```

setRepeat() function will be called when the repeatButton is select/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 select/unselected by the user.it will start/stop the music by calling linkedPlayer.play() / linkedPlay.stop().

when the program tries to play, it will check GUISelector to see if the user 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 measure spinner it changing. to set 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<Interger, Interger> as values. the Pair object content which page is current measure located as key and which line that current measure 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 kind of new GUI component that we created to deliver the ability to config visual elements. when a VElement is selected in GUISelector, 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, 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 storage current select 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

Visulaizer

```
class Visualizer
```

Visualizer is the main class that creates the visual elements, storage them, as well as alignment them.

Also, the visualizer is responsible for providing information about their position.

```
void initMeasures()
```

initMeasures will create the VMeasures object and its sub-nodes reference to the Music XML object (Score).

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

This approach will allow VMeasure and its subobjects store information about its alignment configuration and not lose them during realignment.

```
void alignment()
```

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

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

after change applied, program will try to fit measures in the new lines and pages. also create new mapping of the measures about their poisiton in the page/line.

algorithm that fit measures can be described as 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 this interface. each VconfigAble object has serval internal map to storage following setting:
-current value

-lower/upper limit
-is this configable
-amount to increment/decrement in spinner
There are thoses abstract methods in this class:

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

ConfigAbleList stroge option as key and its current value as value.

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

Limits stroge option as key and its lower/upper limit pair as value.

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

ConfigAble stroge option as key and if this option can be modified by user as value

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

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

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

update current value for given key to the new value.

VElement

```
public class VElement implements VConfigAble
```

VElement is the superclass of all important visual element
following class 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 store its graphic elements in Javafx.group note.

and contain its child element’s group

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

Highlight this VElement and its child element
this implementation is different from VElement to VElement

Color of highlight is determined by enum class HighLight
there is three type of states:
PLAY
SELECTED
NULL
VConfig stores color for each states.

```
public Group getShapeGroups()
```

return group note that contian all graphic element

```
public void alignment()
```

alignment this VElement: put all graphic element 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 ImageResoruce object in a JSON file

ImageResource object contains 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 them on a map. Where 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 music MXL (store as Score object) into music String.
the result 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.

the result music String has two versions:
-the version without repeat (repeat is disabled)
-the version with repeat

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

LinkedPlayer

```
public class LinkedPlayer
```

This class is responsible for playing music with music string.
when creating LinkedPlayer, MXLParser will be called to generate music strings.

Play music
when GUI called the play(measure number) function,
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 repeat.

The pattern is constructed from the specific position to the end of the list.
this position is determined by FirstPosition mapping from MXLParser.

after the pattern is finished, it will be assigned with 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 if repeat is enabled.

now music is ready for playing.
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 player and 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 passed by the LinkedPlayer.

PlayingSelector

```
public class PlayingSelector
```

PlayingSelector is similar to GUISelector. but it will highlight elements with HighLight.Play.
And PlayingSelectror won’t update sidebar meun.

Utility Functions

VUtility

```
class VUtility
```

this class contains help methods that help the visualizer determine position/type/resource

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

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

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

This method get Asset name from given note.
asset name is a string that the program can get an image from the ImageResourceHandler

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

this method will covert note in string into int duration value.
only return denominator

```
public static String getDisplayName(String id)
```

this method will covert internal string id into display name

Vconfig

```
class VConfig implements VConfigAble
```

this class is a singleton class that implements VConfigAble and stores global config 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
-is the repeat enabled.

Maintenance Scenarios

Translating/localization

our project provides an easy way to translate displayed texts.
First, you have to store the language type that the system is currently using in the config. which is the global setting.

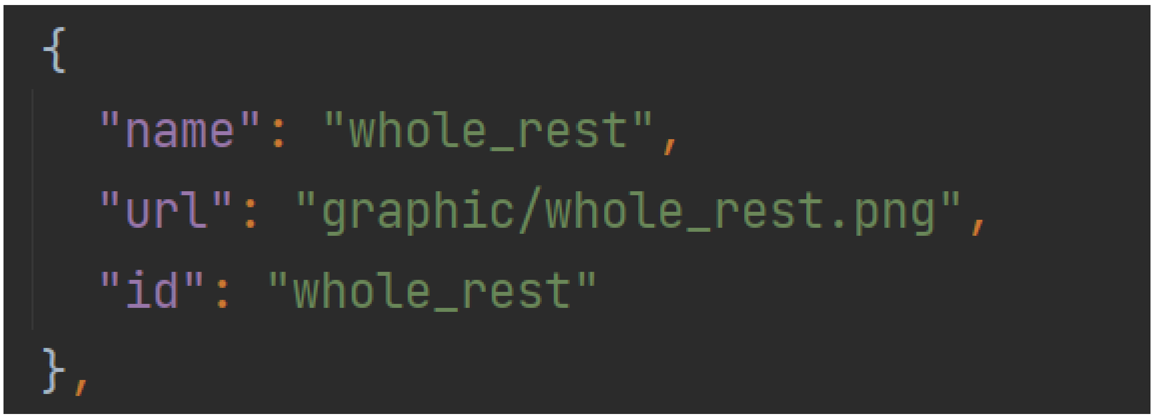
Second, go to VUtility.getDisplayName() and return the translated text with your current language setting.

Notice that the label and text from the Fxml loader won't go through this way. so you have to set it manually.

Edit/Add Image Asset

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

to edit the image asset, find the image asset that you want to change in the imageList.json.
for example, I want to change the whole rest



```
{
  "name": "whole_rest",
  "url": "graphic/whole_rest.png",
  "id": "whole_rest"
},
```

change its URL to your new image asset.
IMPORTANT
your image asset should save in the graphic folder in the format of png file. otherwise, we can’t guarantee you can read that image asset correctly.

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)"
  },
```

Add new VElement

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

1. create your object that extends from VElement
2. initializing the graphic element that you need for this VElement in the constructor
3. override 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 parent node’s group
6. call this Element. alignment in the parent node’s alignment function.

you have added a new VElement!