Design Document

EECS2311 TAB2XML_G14 PROJECT

GROUP 1:

Abdelrahman Altamimi

Hieu Le

Mahdiar Shoraka

Prabjot Dhaliwal

Yongjie Ba

Table of Content

Introduction

This design document is going to describe two different kinds of diagrams for the TAB2XML_G14 project: the class diagram and the sequence diagram. Also, we are going to talk about some scenarios for the maintenance of the project.

Overview

This application functions with the combination of 4 different units: (1) XML parsing unit, (2) Sheet Music Graphics Generation Unit, (3) Music Generation Unit, and (4) View Controller Unit.

1. *XML Parsing Unit*: Works virtually independently of the other 2 units. This Unit parses a music XML file in a string and stores all relevant parsed information within a single Score object.

   Score Class: Contains music XML information using objects of the classes found in the **custom_component_data** package. These classes usually correspond directly with an element/tag name in MusicXML files.
2. *Sheet Music Generation Unit:* Processes the parsed XML data from the XML Parsing Unit, and generates all the appropriate sheet music graphics. The whole sheet music graphics are represented by a single SheetScore object.

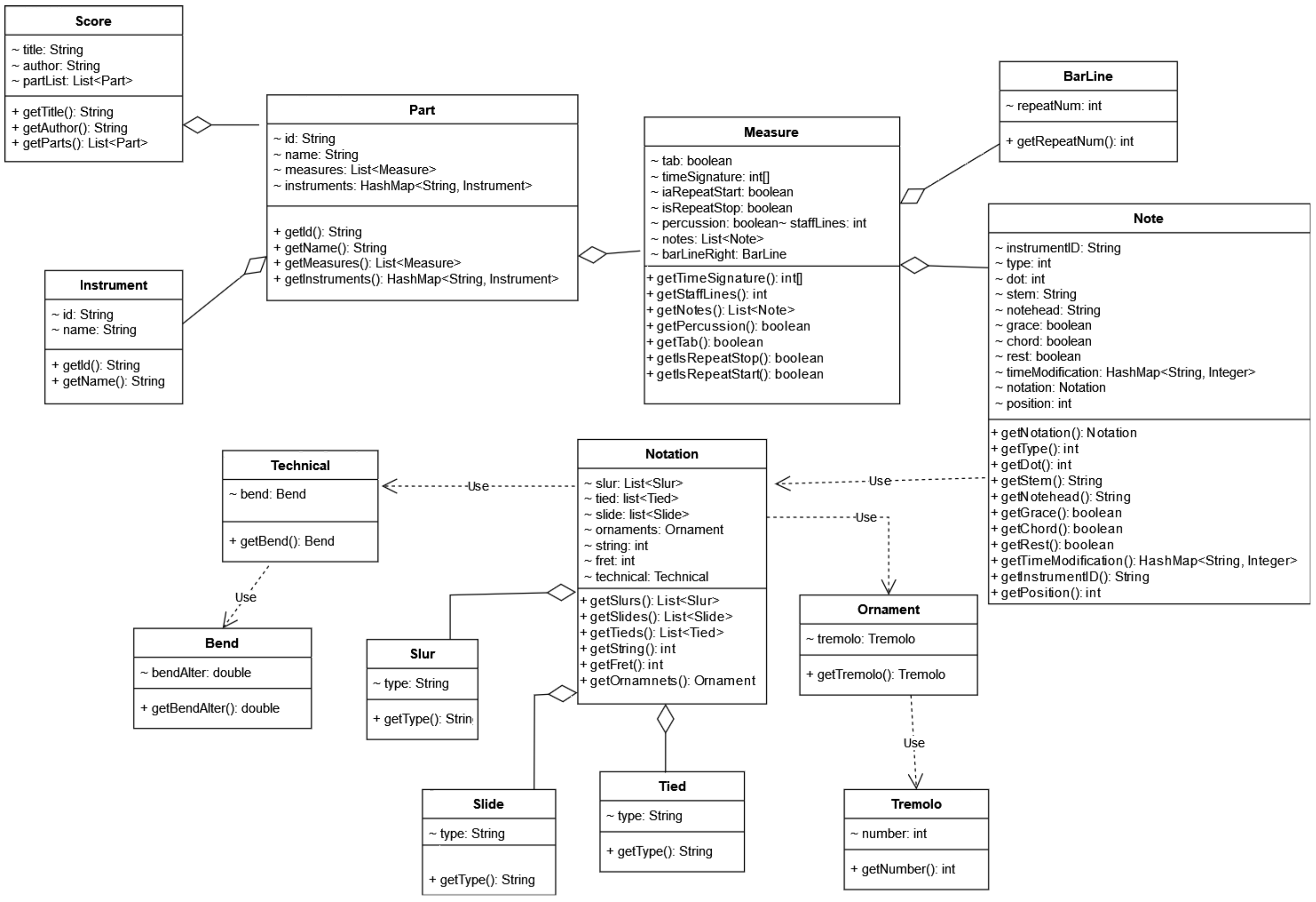
   SheetScore Class: This is a JavaFX graphics element that is composed of other graphical elements of classes found in the **custom_model** package and its sub-packages. Each class is a graphical element (as they extend a JavaFX class)
3. *Music Generation Unit:* Uses the Score object from the XML parsing unit to develop a JFugue pattern string. This audio is played via this string using the JFugue library. Handles all audio playback requests. This unit is composed of the single **music player** class within the **customer_player** package.
4. *View Controller Unit:* Links all the units, to create a GUI where input is received as tablature and the following are outputted accordingly with user interactions: sheet music graphics, audio playback, PDF exporting, and customizability settings. This unit is

composed of classes found in the **GUI** package (classes of interest are shown on the next page)

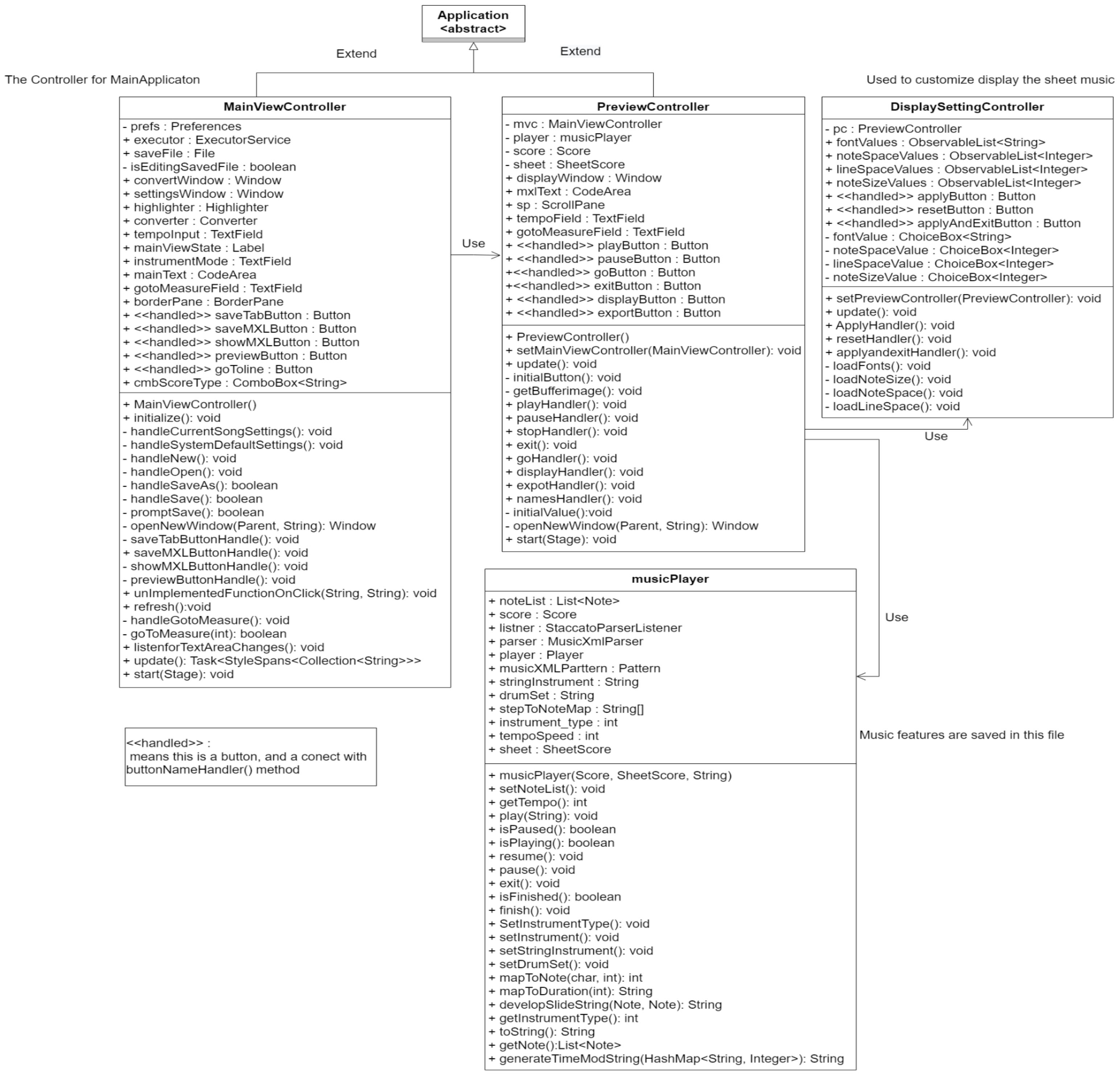
Class Diagram

Notice: Because adding all the classes into one diagram may lead to a crowded and hard-to-read diagram, thus This document has separated them into several class diagrams

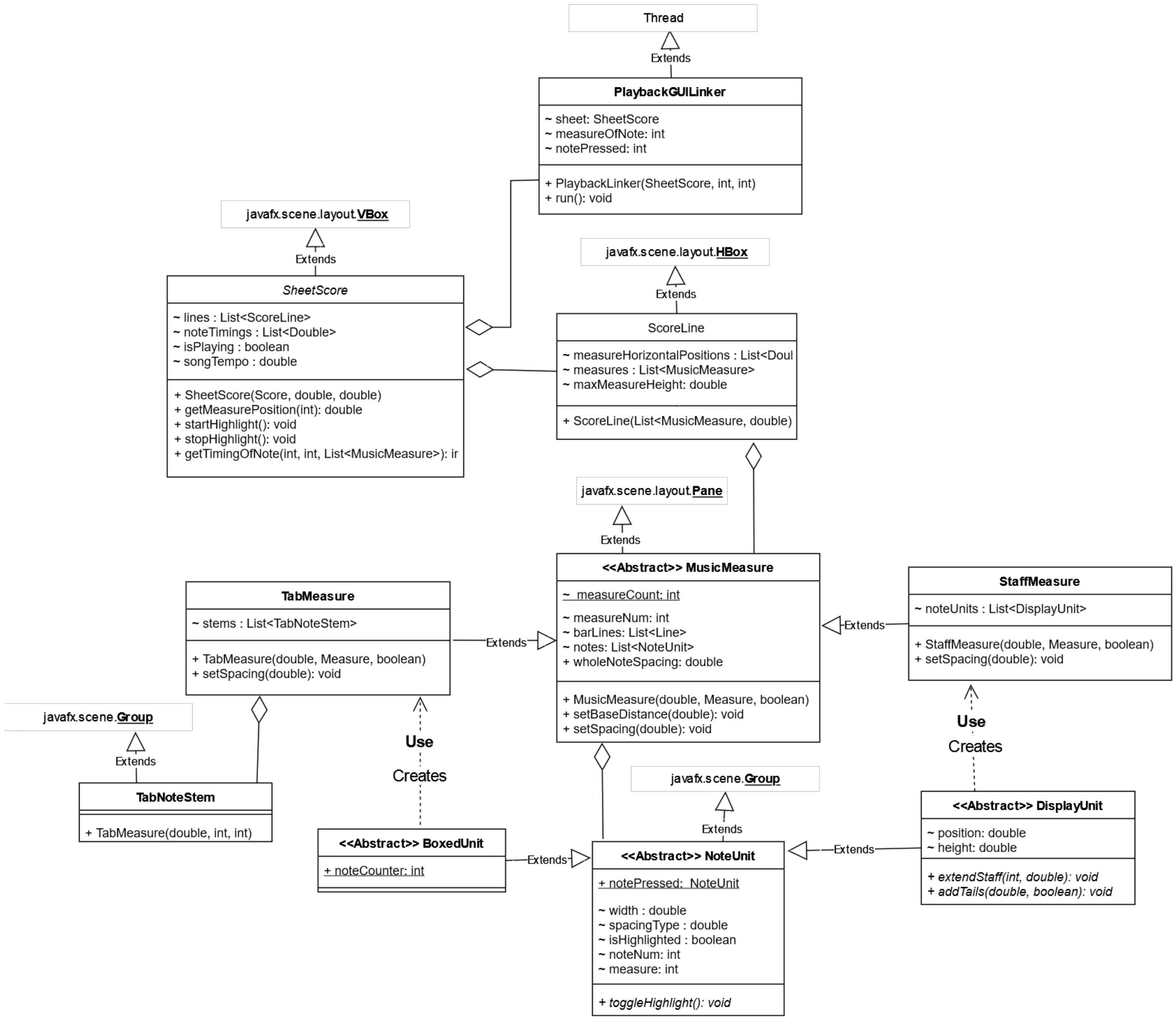
Music-XML Parsing Unit:



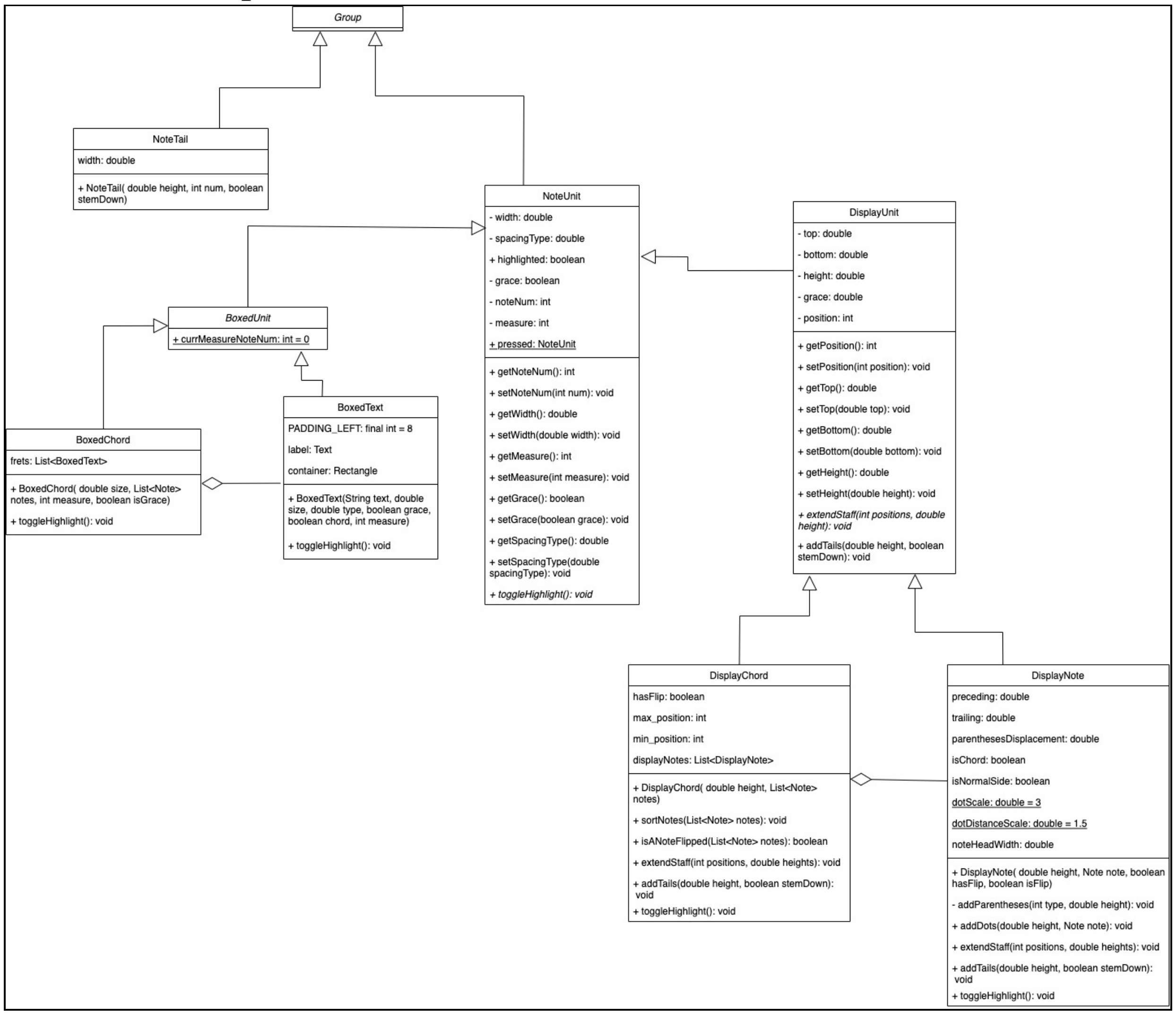
Music Generation/View Controller Unit Class Diagram


Application
<abstract>
Extend
Extend
The Controller for MainApplicaton
Used to customize display the sheet music
MainViewController
- prefs : Preferences
+ executor : ExecutorService
+ saveFile : File
- isEditingSavedFile : boolean
+ convertWindow : Window
+ settingsWindow : Window
+ highlighter : Highlighter
+ converter : Converter
+ tempoInput : TextField
+ mainViewState : Label
+ instrumentMode : TextField
+ mainText : CodeArea
+ gotoMeasureField : TextField
+ borderPane : BorderPane
+ <<handled>> saveTabButton : Button
+ <<handled>> saveMXLButton : Button
+ <<handled>> showMXLButton : Button
+ <<handled>> previewButton : Button
+ <<handled>> goToline : Button
+ cmbScoreType : ComboBox<String>
+ MainViewController()
+ initialize(): void
- handleCurrentSongSettings(): void
- handleSystemDefaultSettings(): void
- handleNew(): void
- handleOpen(): void
- handleSaveAs(): boolean
- handleSave(): boolean
- promptSave(): boolean
- openNewWindow(Parent, String): Window
- saveTabButtonHandle(): void
+ saveMXLButtonHandle(): void
- showMXLButtonHandle(): void
- previewButtonHandle(): void
+ unImplementedFunctionOnClick(String, String): void
+ refresh():void
- handleGotoMeasure(): void
- goToMeasure(int): boolean
+ listenforTextAreaChanges(): void
+ update(): Task<StyleSpans<Collection<String>>>
+ start(Stage): void
Use
PreviewController
- mvc : MainViewController
- player : musicPlayer
- score : Score
- sheet : SheetScore
+ displayWindow : Window
+ mxlText : CodeArea
+ sp : ScrollPane
+ tempoField : TextField
+ gotoMeasureField : TextField
+ <<handled>> playButton : Button
+ <<handled>> pauseButton : Button
+<<handled>> goButton : Button
+<<handled>> exitButton : Button
+ <<handled>> displayButton : Button
+ <<handled>> exportButton : Button
+ PreviewController()
+ setMainViewController(MainViewController): void
+ update(): void
- initialButton(): void
- getBufferimage(): void
+ playHandler(): void
+ pauseHandler(): void
+ stopHandler(): void
+ exit(): void
+ goHandler(): void
+ displayHandler(): void
+ expotHandler(): void
+ namesHandler(): void
- initialValue():void
- openNewWindow(Parent, String): Window
+ start(Stage): void
DisplaySettingController
- pc : PreviewController
+ fontValues : ObservableList<String>
+ noteSpaceValues : ObservableList<Integer>
+ lineSpaceValues : ObservableList<Integer>
+ noteSizeValues : ObservableList<Integer>
+ <<handled>> applyButton : Button
+ <<handled>> resetButton : Button
+ <<handled>> applyAndExitButton : Button
- fontValue : ChoiceBox<String>
- noteSpaceValue : ChoiceBox<Integer>
- lineSpaceValue : ChoiceBox<Integer>
- noteSizeValue : ChoiceBox<Integer>
+ setPreviewController(PreviewController): void
+ update(): void
+ ApplyHandler(): void
+ resetHandler(): void
+ applyandexitHandler(): void
- loadFonts(): void
- loadNoteSize(): void
- loadNoteSpace(): void
- loadLineSpace(): void
Use
Use
musicPlayer
+ noteList : List<Note>
+ score : Score
+ listner : StaccatoParserListener
+ parser : MusicXmlParser
+ player : Player
+ musicXMLPartten : Pattern
+ stringInstrument : String
+ drumSet : String
+ stepToNoteMap : String[]
+ instrument_type : int
+ tempoSpeed : int
+ sheet : SheetScore
+ musicPlayer(Score, SheetScore, String)
+ setNoteList(): void
+ getTempo(): int
+ play(String): void
+ isPaused(): boolean
+ isPlaying(): boolean
+ resume(): void
+ pause(): void
+ exit(): void
+ isFinished(): boolean
+ finish(): void
+ SetInstrumentType(): void
+ setInstrument(): void
+ setStringInstrument(): void
+ setDrumSet(): void
+ mapToNote(char, int): int
+ mapToDuration(int): String
+ developSlideString(Note, Note): String
+ getInstrumentType(): int
+ toString(): String
+ getNote():List<Note>
+ generateTimeModString(HashMap<String, Integer>): String
Music features are saved in this file
<<handled>> :
means this is a button, and a conect with
buttonNameHandler() method

High-Level Sheet Music Graphics Generation:

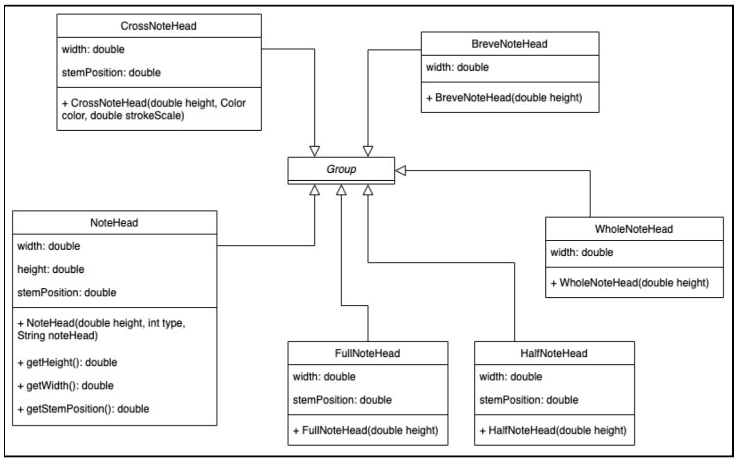

Thread
Extends
PlaybackGUILinker
~ sheet: SheetScore
~ measureOfNote: int
~ notePressed: int
+ PlaybackLinker(SheetScore, int, int)
+ run(): void
javafx.scene.layout.VBox
Extends
SheetScore
~ lines : List<ScoreLine>
~ noteTimings : List<Double>
~ isPlaying : boolean
~ songTempo : double
+ SheetScore(Score, double, double)
+ getMeasurePosition(int): double
+ startHighlight(): void
+ stopHighlight(): void
+ getTimingOfNote(int, int, List<MusicMeasure>): ir
javafx.scene.layout.HBox
Extends
ScoreLine
~ measureHorizontalPositions : List<Doul
~ measures : List<MusicMeasure>
~ maxMeasureHeight: double
+ ScoreLine(List<MusicMeasure, double)
javafx.scene.layout.Pane
Extends
<<Abstract>> MusicMeasure
~ measureCount: int
~ measureNum: int
~ barLines: List<Line>
~ notes: List<NoteUnit>
+ wholeNoteSpacing: double
+ MusicMeasure(double, Measure, boolean)
+ setBaseDistance(double): void
+ setSpacing(double): void
TabMeasure
~ stems : List<TabNoteStem>
+ TabMeasure(double, Measure, boolean)
+ setSpacing(double): void
Extends
StaffMeasure
~ noteUnits : List<DisplayUnit>
+ StaffMeasure(double, Measure, boolean)
+ setSpacing(double): void
Extends
javafx.scene.Group
Extends
TabNoteStem
+ TabMeasure(double, int, int)
Use
Creates
<<Abstract>> BoxedUnit
+ noteCounter: int
Extends
javafx.scene.Group
Extends
<<Abstract>> NoteUnit
+ notePressed: NoteUnit
~ width : double
~ spacingType : double
~ isHighlighted : boolean
~ noteNum: int
~ measure: int
+ toggleHighlight(): void
Use
Creates
<<Abstract>> DisplayUnit
~ position: double
~ height: double
+ extendStaff(int, double): void
+ addTails(double, boolean): void
Extends

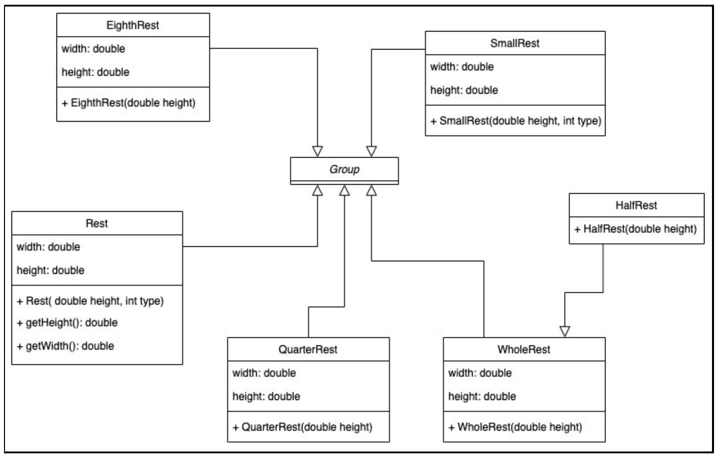
Music Note Graphics Generation:


Group
NoteTail
width: double
+ NoteTail(double height, int num, boolean stemDown)
NoteUnit
- width: double
- spacingType: double
+ highlighted: boolean
- grace: boolean
- noteNum: int
- measure: int
+ pressed: NoteUnit
+ getNoteNum(): int
+ setNoteNum(int num): void
+ getWidth(): double
+ setWidth(double width): void
+ getMeasure(): int
+ setMeasure(int measure): void
+ getGrace(): boolean
+ setGrace(boolean grace): void
+ getSpacingType(): double
+ setSpacingType(double spacingType): void
+ toggleHighlight(): void
DisplayUnit
- top: double
- bottom: double
- height: double
- grace: double
- position: int
+ getPosition(): int
+ setPosition(int position): void
+ getTop(): double
+ setTop(double top): void
+ getBottom(): double
+ setBottom(double bottom): void
+ getHeight(): double
+ setHeight(double height): void
+ extendStaff(int positions, double height): void
+ addTails(double height, boolean stemDown): void
BoxedUnit
+ currMeasureNoteNum: int = 0
BoxedText
PADDING_LEFT: final int = 8
label: Text
container: Rectangle
+ BoxedText(String text, double size, double type, boolean grace, boolean chord, int measure)
+ toggleHighlight(): void
BoxedChord
frets: List<BoxedText>
+ BoxedChord(double size, List<Note> notes, int measure, boolean isGrace)
+ toggleHighlight(): void
DisplayChord
hasFlip: boolean
max_position: int
min_position: int
displayNotes: List<DisplayNote>
+ DisplayChord(double height, List<Note> notes)
+ sortNotes(List<Note> notes): void
+ isANoteFlipped(List<Note> notes): boolean
+ extendStaff(int positions, double heights): void
+ addTails(double height, boolean stemDown): void
+ toggleHighlight(): void
DisplayNote
preceding: double
trailing: double
parenthesesDisplacement: double
isChord: boolean
isNormalSide: boolean
dotScale: double = 3
dotDistanceScale: double = 1.5
noteHeadWidth: double
+ DisplayNote(double height, Note note, boolean hasFlip, boolean isFlip)
- addParentheses(int type, double height): void
+ addDots(double height, Note note): void
+ extendStaff(int positions, double heights): void
+ addTails(double height, boolean stemDown): void
+ toggleHighlight(): void

The following 2 Diagrams are GUI elements used by the DisplayNote class:

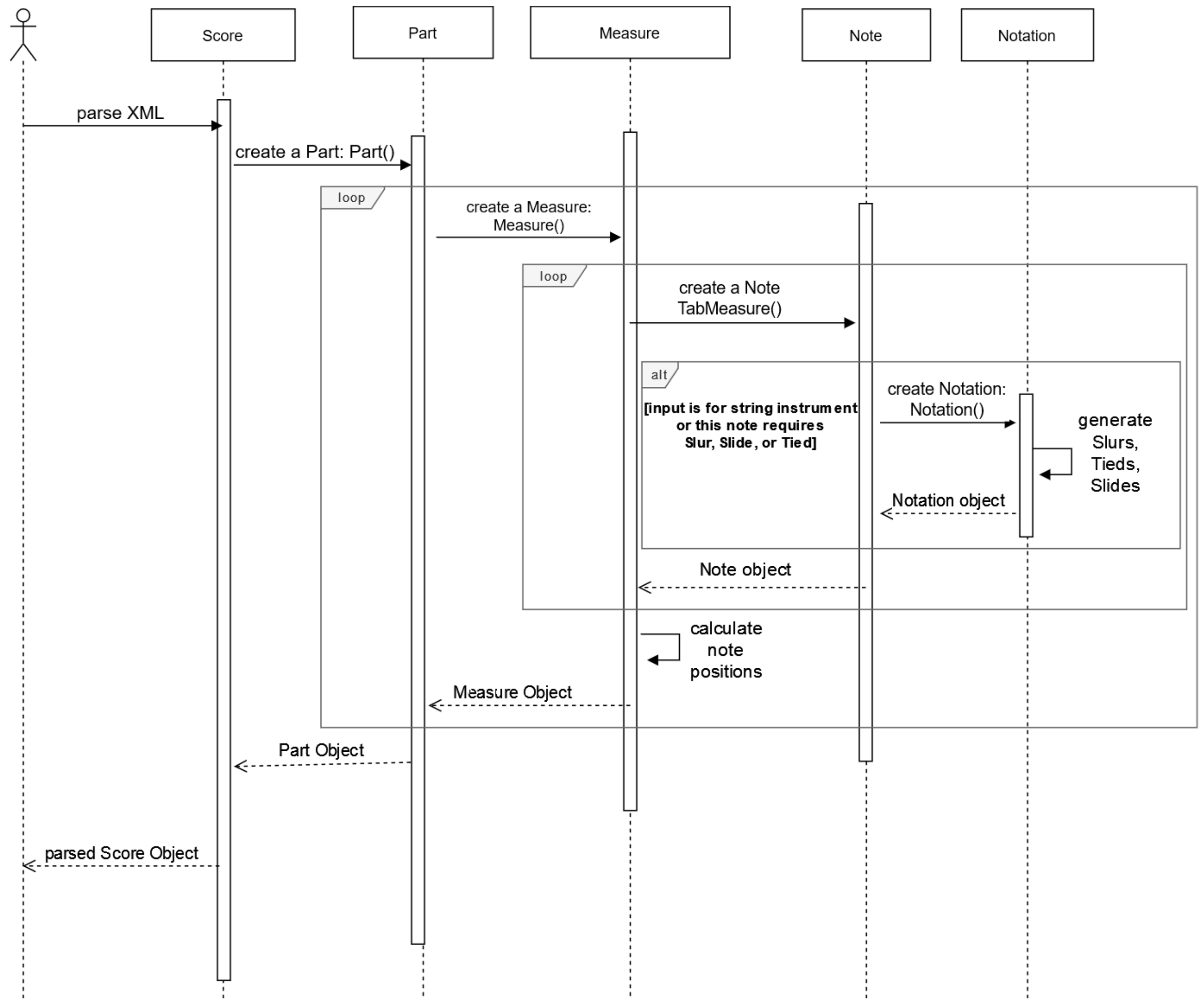
(Notehead) **(Rest)**



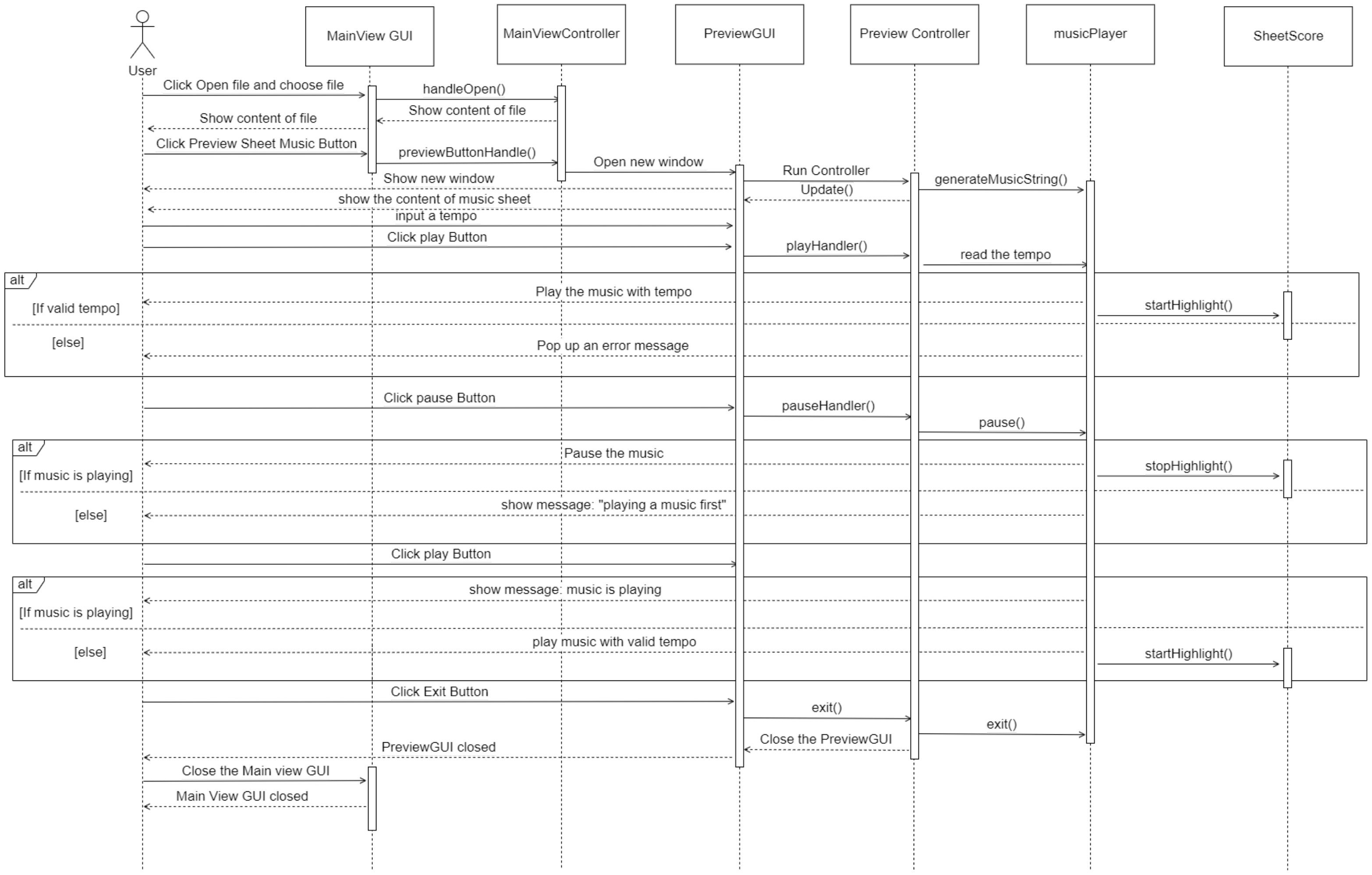


Sequence Diagram

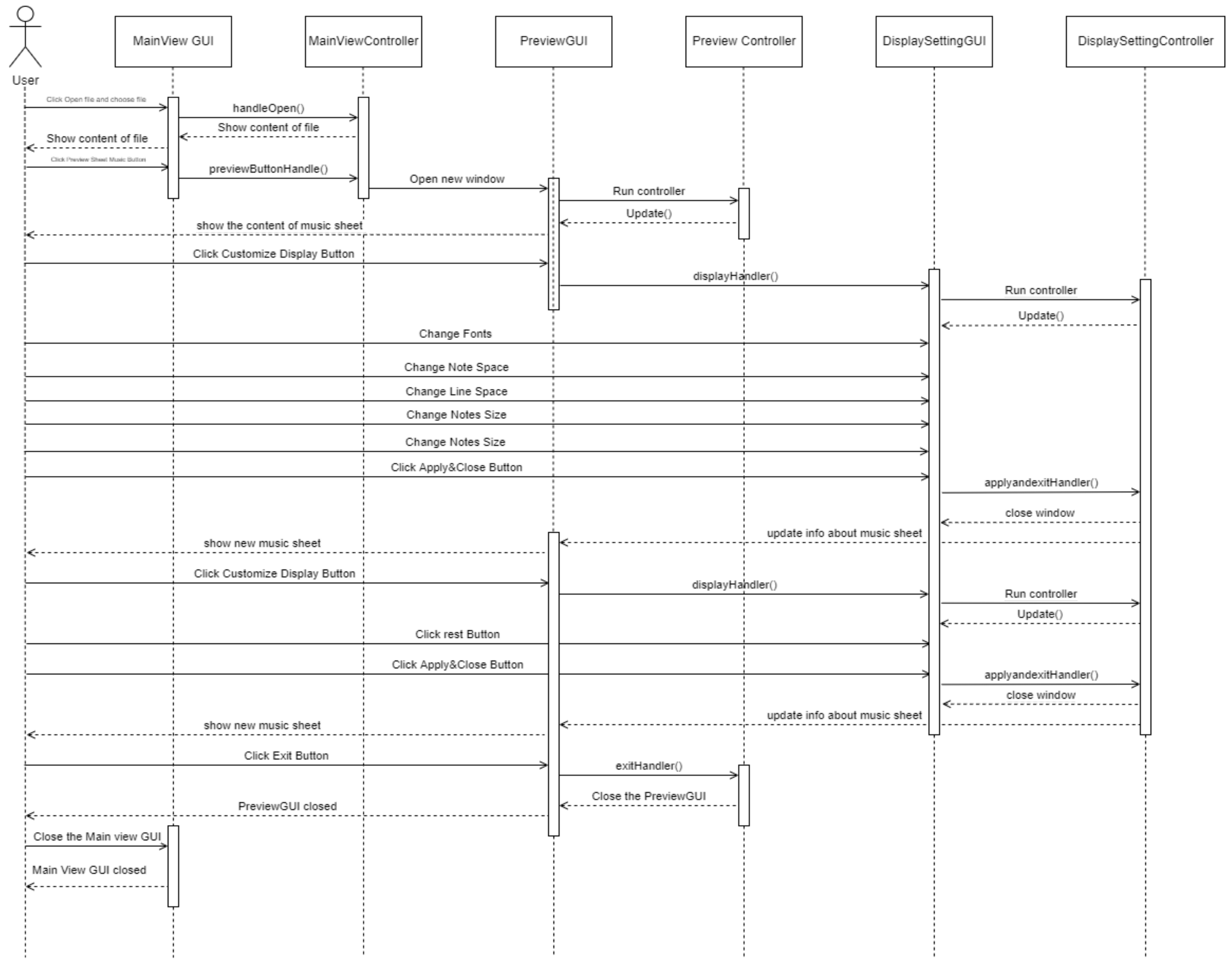
Music-XML Parsing Unit


Score
Part
Measure
Note
Notation
parse XML
create a Part: Part()
loop
create a Measure: Measure()
loop
create a Note TabMeasure()
alt
[input is for string instrument or this note requires Slur, Slide, or Tied]
create Notation: Notation()
generate Slurs, Tieds, Slides
Notation object
Note object
calculate note positions
Measure Object
Part Object
parsed Score Object

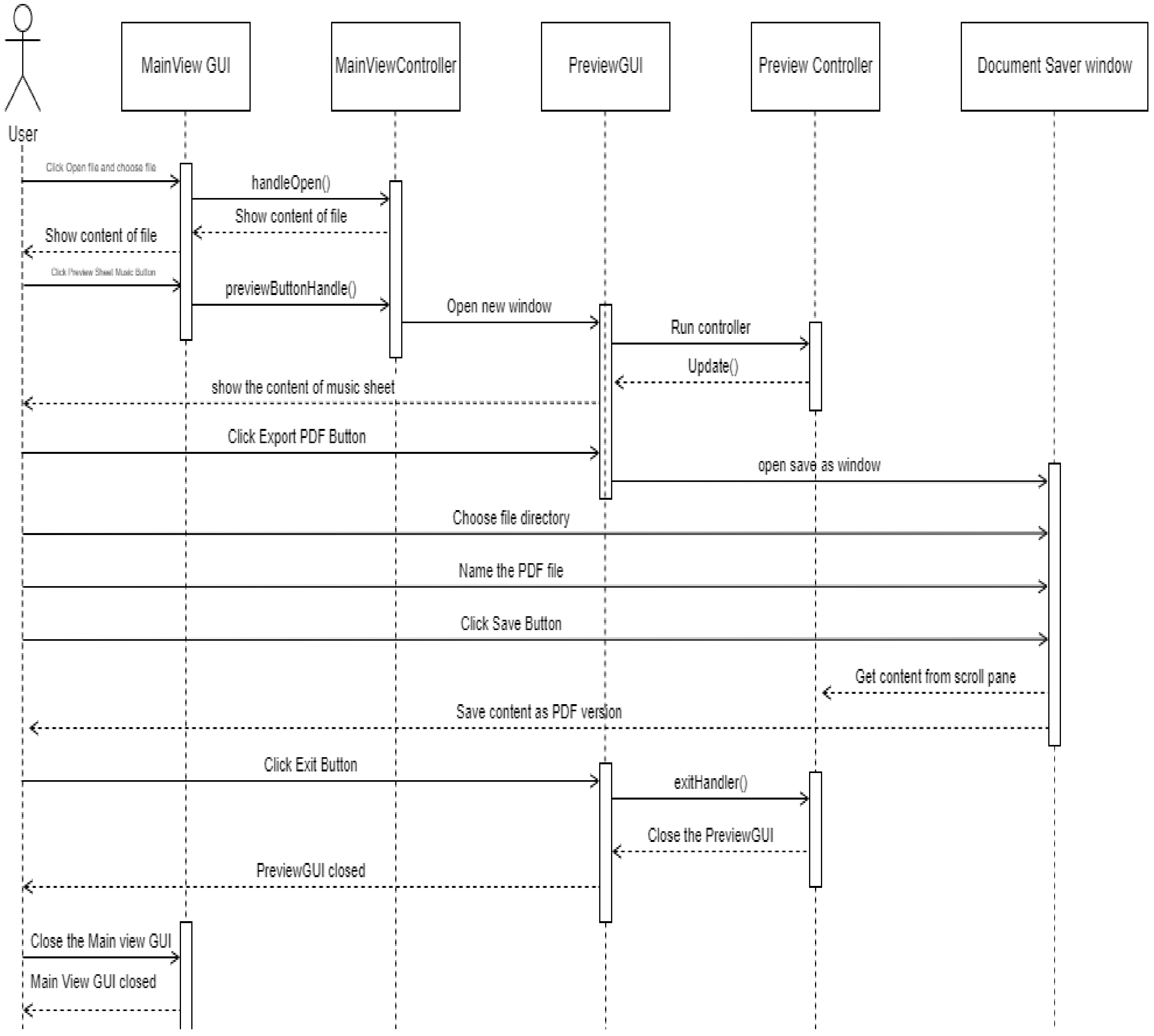
Music Generation Unit / Note Highlighting Unit



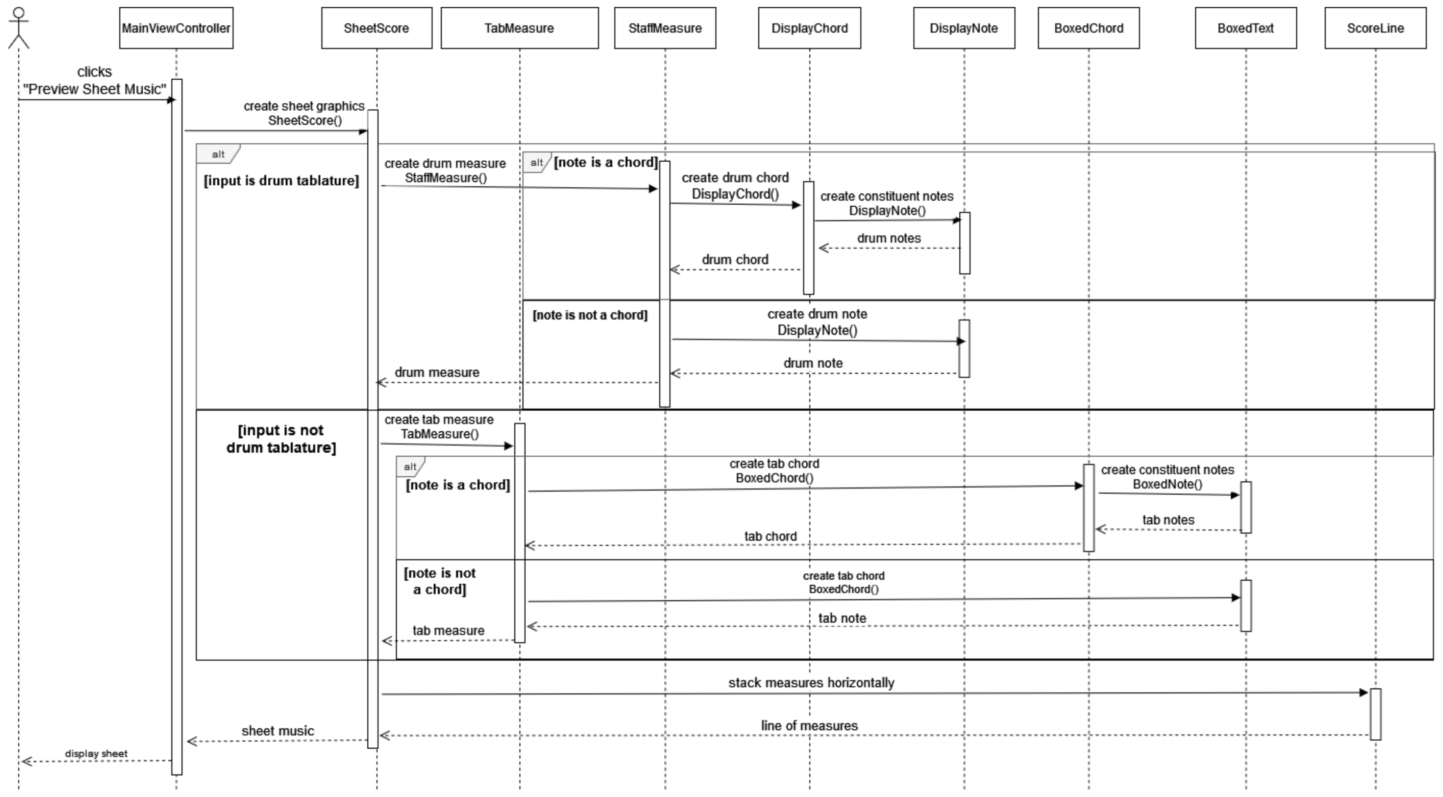
Customize Display



Export PDF


User
MainView GUI
MainViewController
PreviewGUI
Preview Controller
Document Saver window
Click Open file and choose file
handleOpen()
Show content of file
Show content of file
Click Preview Sheet Music Button
previewButtonHandle()
Open new window
Run controller
Update()
show the content of music sheet
Click Export PDF Button
open save as window
Choose file directory
Name the PDF file
Click Save Button
Get content from scroll pane
Save content as PDF version
Click Exit Button
exitHandler()
Close the PreviewGUI
PreviewGUI closed
Close the Main view GUI
Main View GUI closed

Generation of Sheet Music Graphics:


MainViewController
SheetScore
TabMeasure
StaffMeasure
DisplayChord
DisplayNote
BoxedChord
BoxedText
ScoreLine
clicks "Preview Sheet Music"
create sheet graphics SheetScore()
alt
[input is drum tablature]
create drum measure StaffMeasure()
alt [note is a chord]
create drum chord DisplayChord()
create constituent notes DisplayNote()
drum notes
drum chord
[note is not a chord]
create drum note DisplayNote()
drum note
drum measure
[input is not drum tablature]
create tab measure TabMeasure()
alt
[note is a chord]
create tab chord BoxedChord()
create constituent notes BoxedNote()
tab notes
tab chord
[note is not a chord]
create tab chord BoxedChord()
tab note
tab measure
stack measures horizontally
line of measures
sheet music
display sheet

Maintenance Scenarios

I. Play function scenarios

All play features are included in the musicPlayer.java file which is saved in the custom_player package.

A. If we want to add a new notification about playing music, only need to add a new notification in the musicPlayer.java class.

B. If we want to change the condition for the play, pause or exit feature, only need to change musicPlayer.java class.

C. If we want to support new features like playing music from the measure that the user input or the notion user clicked, also the musicPlayer.java is the only class that needs to be changed.

D. If we want to support more instruments such as piano, celesta, clavinet and so on, we only need to change musicPlayer.java class.

II. Modification of XML Parsing Unit

A. (1) If a new MusicXML element must be supported, simply create a new class ***X*** which has the attributes corresponding to the content/attributes of this element. (2) Then, create an object of this element within the corresponding class for the new element's parent. (3) Make sure to store this object within the class it is initialized in, so it can be referenced via an object of this class. (4) The initialization of the attributes of class ***X*** must be handled by the constructor the class itself which has the single parameter(**doc. Element)** containing the XML element in the tag: <new-supported-element-tag-name>.

B. If previously unsupported attributes/contents of music XML elements must be supported, then add these attributes to the corresponding Java class in the custom_component_data package. Make sure to initialize them within the class's constructor.

III. Updating the Music Sheet Graphics Generation

A. If a new graphical music element must be supported, and this element applies to musical notes, then follow the steps below:

1. Create a new class simply generating the graphical component without any translations. Make sure its constructor has the appropriate arguments and it extends a JavaFX class.
2. Set and initialize relevant data that may be used (such as its width, or its enumerated occurrence within a measure)
3. Accordingly, initialize an object of this note using appropriate parameters within the DisplayNote or DisplayChord class for drums, and the BoxedText or BoxedChord class for guitars. Then add this object to the display by "this.getChildren().add(element);" for the new element created.
4. Translate the element horizontally and horizontally based on its position on the note (The top-left corner of the square around the notehead is the origin (0,0)).
5. Modify the relevant attributes of the note (if this new element causes the note to take up a greater width, then increment the width accordingly). Such values can be "spacing type, width, preceding, trailing" which are further documented in the project code.