ALLEGRO TAB CONVERTER

DESIGN DOCUMENT

Written by:
Mohammed Fulwala
Junhyeong Park
Rafael Dolores
Shawn Verma
Yashraj Rathore

TABLE OF CONTENTS

Introduction

This document is for the software system, Allegro Tab Converter, which converts a guitar, drum, or bass tex tablature into a musicXML file, which can be used to play in various software (eg: MuseScore) and websites (eg: SoundSlice). This software was created because there is no simple way to convert a text tablature for an instrument into the desired file type, which is the musicXML file. This document provides a breakdown of the system design at various levels of the project.

System Overview

Software Design

The software system follows a MVC design structure. This structure provides a simplistic design for a user application. The MVC structure allows for software division between the separate components of the system. For this app we model the XML data structure.

The technologies used to assist the creation of the project was JacksonXML, JavaFX, and TestFX.

System Architecture

This section explains the entire architecture of the internal part of this software system, which is basically the parser. We shall first start with the guitar and bass guitar. The guitar is one instrument, which has six strings, with each string producing a different tune. As seen in the image below, we have a bunch of classes called Parts, Score Partwise, and many others. MusicXML is a type of xml file, and we need certain components of the xml file, which is why we created these classes. They are nothing more but models. We need these models because we are using a library called JacksonXML, which can be used to generate XML files as long as the models are being created for the components of the XML file.

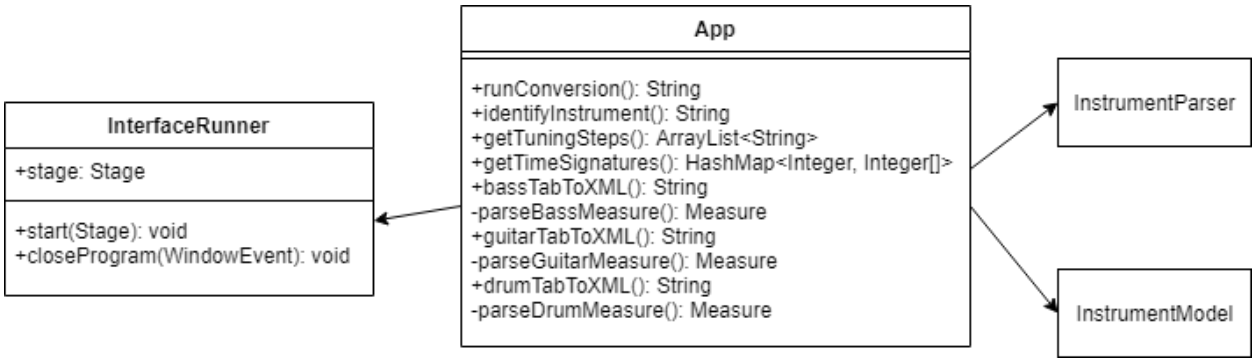
Now, these classes that are referred to as models contain fields, and their getter and setter methods. They are the elements for the XML tags. For example, as shown below, the pitch tag consists of step, and octave.

```
<pitch>
  <step>E</step>
  <octave>2</octave>
  </pitch>
```

If you look at the class diagram below, you will notice that the class Pitch has two fields, step and octave, which are Strings, and their getter and setter methods (which have not been shown due to space limitations). You can

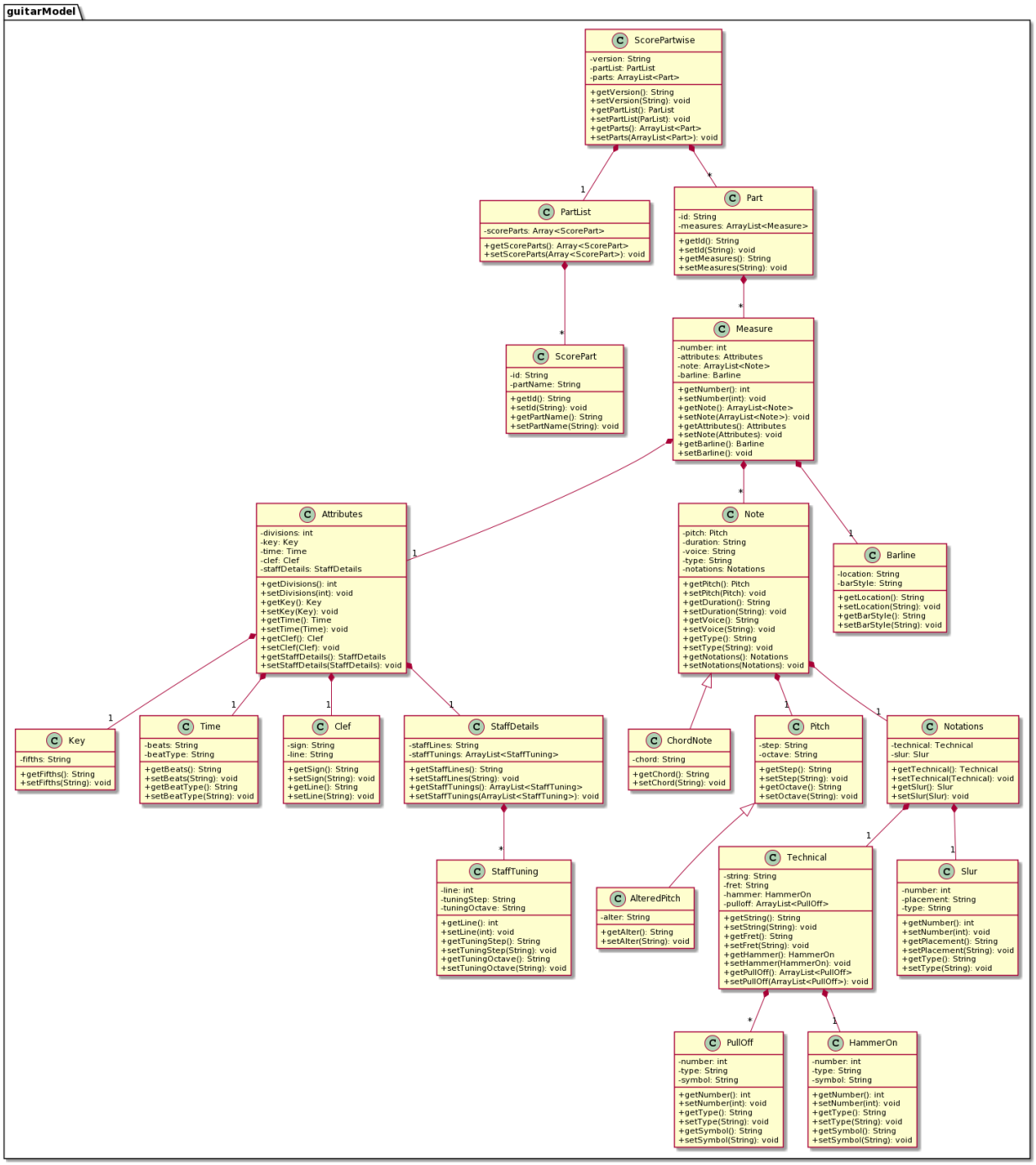
assume that for every field shown in the diagram below, we have their getter and setter methods as well.

The control of the software is the class App.java. This class is responsible for interacting between the user interface and the backend. The app runs the conversion from the user input, considering any customizations given. The app determines the instrument of the input tablature and converts the tablature by use of helper classes called InstrumentParser. The instrument parser contains the relevant methods to getting the attributes of the tablature for the specified instrument type. This data is then converted into an instrument model. This instrument model is then converted into an XML file using an XML mapper. This is the general flow of the logic that occurs for the conversion function of the app.



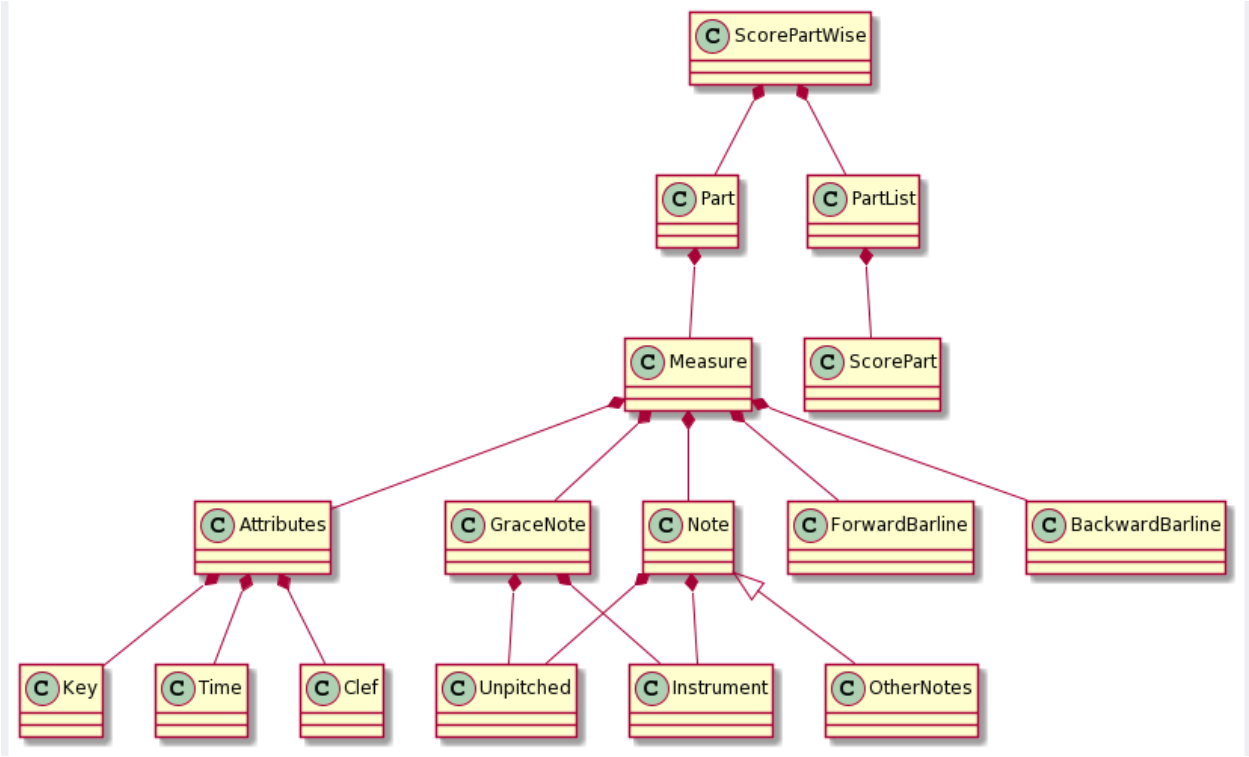
If you look at the class diagram below, you will notice that the class Pitch has two fields, step and octave, which are Strings, and their getter and setter methods (which

have not been shown due to space limitations). You can assume that for every field shown in the diagram below, we have their getter and setter methods as well.



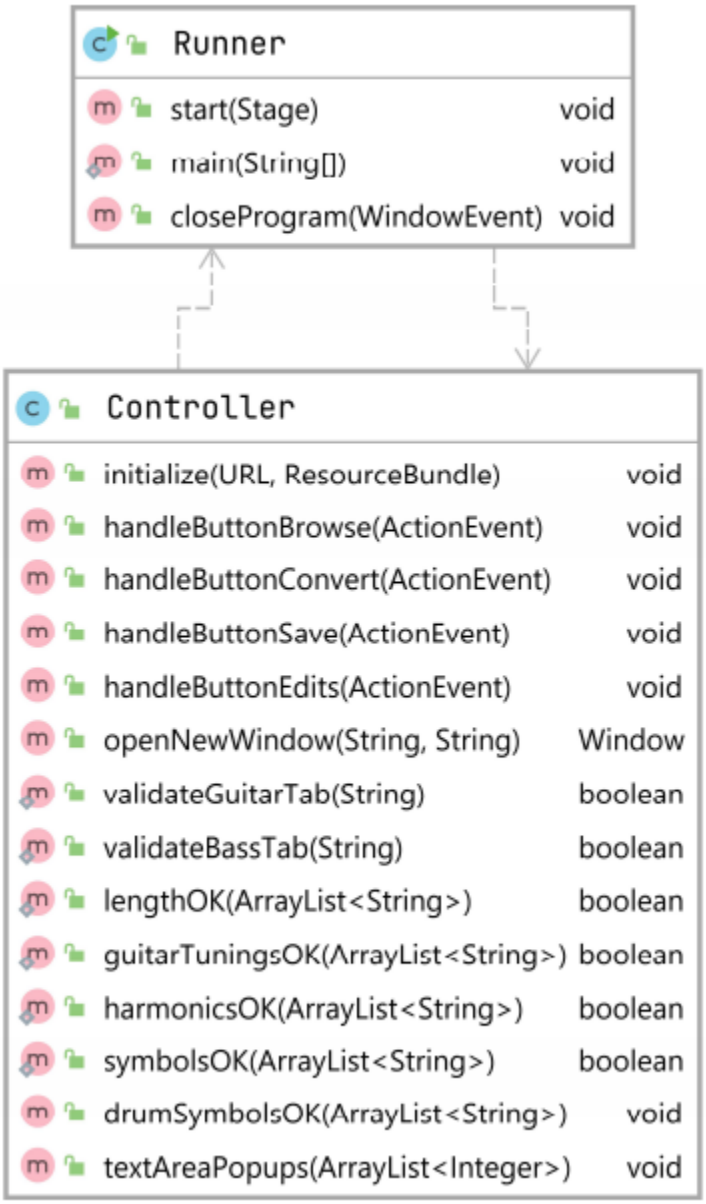
The bass guitar xml file has the same components as the guitar xml file, so this document will not go in detail about it. The guitar and the bass guitar are the same, except for a couple of minor differences, such as the octave of the notes, and the number of strings in the bass. The music theory is a bit different, but otherwise the same model can be used for the bass as well. For this software system, we have created a different package for the Bass Model. The purpose is so that we can add any feature that is unique to the Bass only in the future, if we decide to do so.

The Drums is an interesting case, since it consists of most of the features of the guitar. However, unlike the Bass Guitar, it has major differences in its xml file. As mentioned above, the guitar has six strings, each string with a different tune. For the drum, we have multiple instruments combined together, which makes it complicated. The diagram shown below consists of the extra classes that must be implemented in the model in order to construct the xml file.



These classes on their own won't be able to generate the XML file, which is why we need more classes. In these classes, we will create objects of these classes and map them together to form an xml file. All of this is possible because of the JacksonXML library.

HUMAN INTERFACE DESIGN



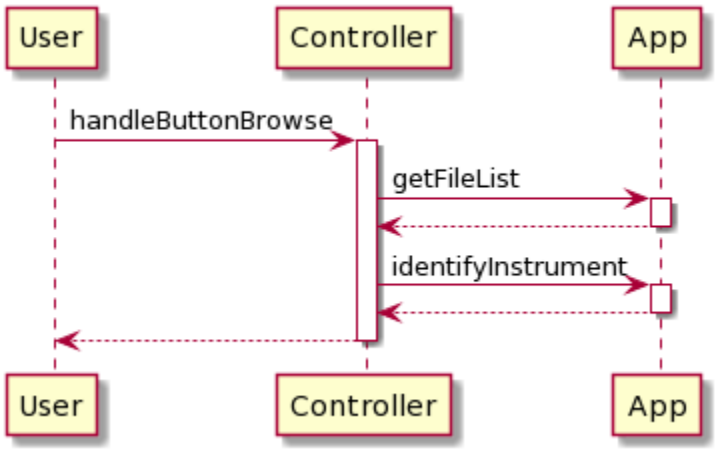
The Graphic User Interface (GUI) for this system has been designed using the JavaFX library. There are classes as shown below.

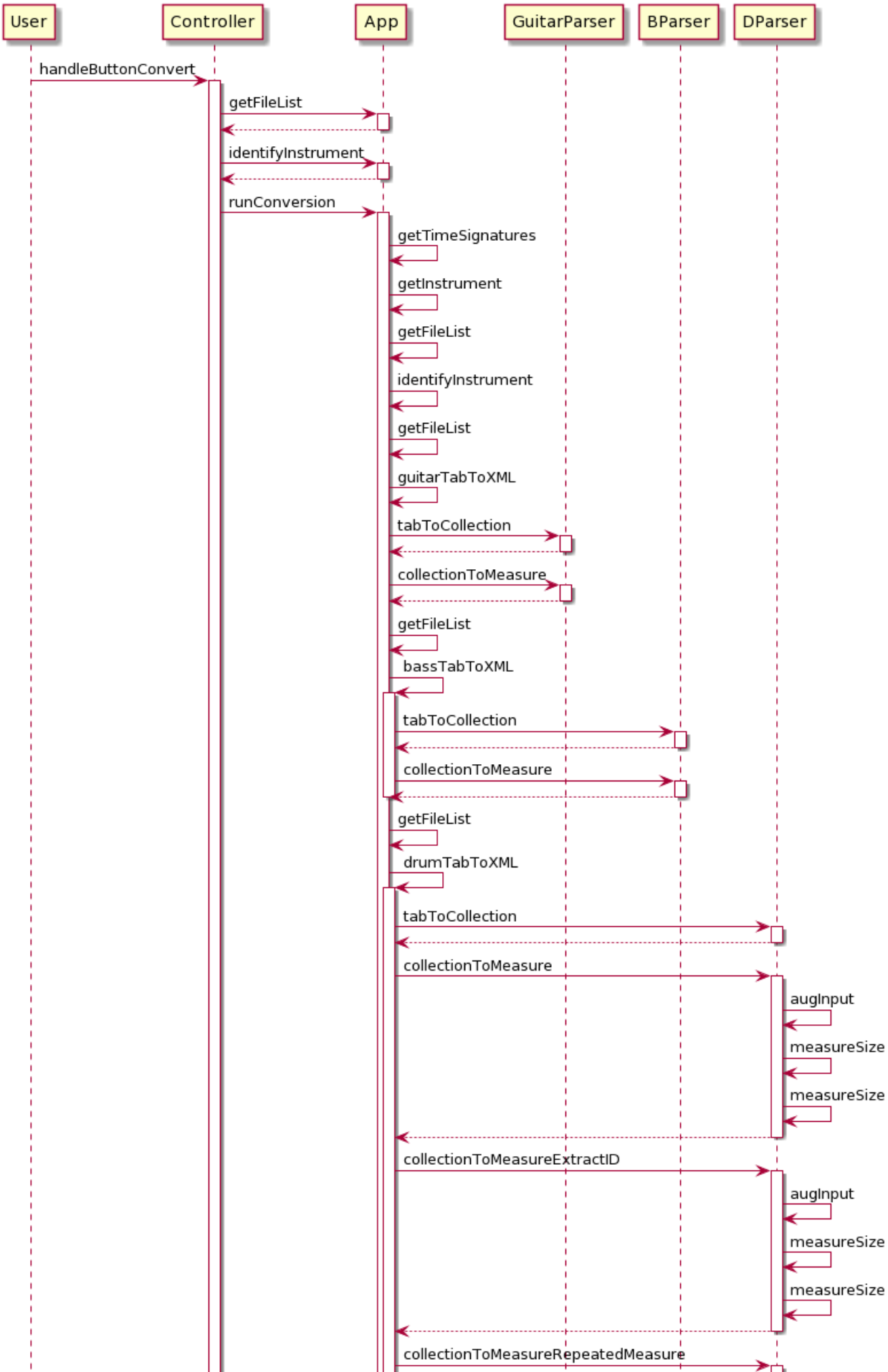
The Runner class is the main class that launches the application. In this class, we decide the size of the application, and attach the FXML file as well as the CSS file to the stage of the application.

The Controller class is the class where each and every component of the GUI is defined as fields, and the methods in this class handles every action event of the application. An action event is basically any event that occurs by the actions of the user. During these action events, a specific series of actions are implemented by a click of a button, or by typing something on a textbox.

The GUI has two text boxes, one where the user can paste a tablature if they do not have a file. It is also a place where the user can edit any note or add something in the tablature. The second textbox cannot be edited, and is there for the user to be able to view the generated XML file. The system consists of four buttons: Browse File, Convert, Save and Export. The save button allows the user to save any edits made on the text tablature.

A diagram showing how the GUI is connected with the Parser classes, and how it helps in identifying the instrument. It also shows what methods are called when the convert button is pressed.




User
Controller
App
GuitarParser
BParser
DParser
handleButtonConvert
getFileList
identifyInstrument
runConversion
getTimeSignatures
getInstrument
getFileList
identifyInstrument
getFileList
guitarTabToXML
tabToCollection
collectionToMeasure
getFileList
bassTabToXML
tabToCollection
collectionToMeasure
getFileList
drumTabToXML
tabToCollection
collectionToMeasure
augInput
measureSize
measureSize
collectionToMeasureExtractID
augInput
measureSize
measureSize
collectionToMeasureRepeatedMeasure

Please click here to view it in better resolution: http://www.plantuml.com/plantuml/png/rPNFIWD13CRlynJx0dq13z8_LIcq87O5hsadhWExCqicAUhJErkx4d1mAFHWBuNq-_8blynXfv61MBhMz43izgu7pzLpH3B3JpMvhHR1CuMsJTLDzGRUjZWNvk2JS41Y0tjs1s1KJgEQKiIixrDlWtplMboxg7gIP0PveYUHiQWTsJXdqRDx_LZvo2GT-ekaafoueWi-4wXWQjVXrZKUM0ZZBrdjfILz1YKeeBCR6t4CLCEk3Y-RzQWz7Fzz18h7b-D1NWoqYKcZQP_HnV5w0xZ5l3a_QFypemuRX2Y4bnfMFiKEOXmFaNszUV4WeyNCyvjfYmujmvyHR36pV0dUadH_mY-ByAF5BFCxAeiABSDdWdKdIBFolN1UxRwpjkuJouOBBNxtpfGQLykh9NZ27e7HdighvvXrGJp_Nmmjd_CL8cGXRwRehNJj5m00

Maintenance Scenarios:

(a) Supporting A New Instrument

(i) Classes/ Folders to be added:

- A new model package for the new instrument
- A new parser class for the instrument under TAB_TO_XML folder

(ii) Classes to be modified:

- App.java

(iii) Details of the modifications to be performed

- Add a new test condition for identifyInstrument that pertains to the new instrument.
- Add a separate folder for the instrument and all the necessary classes.
- Inside those classes, declare the necessary fields along with the constructor, setters, and getters.
- For app.java, make two new methods, newInstrumentToXml that accepts an arraylist of strings (which represents the content of the inputted text-tab) and parseNewInstrumentMeasure that will parse each measure sequentially.

(b) Supporting New Note Techniques

(i) Classes / Folders to be added:

- New class to represent the technique on the model of the instrument where it belongs too.
- New classes to represent the possible attributes of the new technique

(ii) Classes to be modified:

- App.java

(iii) Details of the modifications to be performed:

- Create the necessary classes inside of the instrument model
- Add any necessary fields inside those classes and create its getters and setters.
- Go to app.java and inside of the parseMeasure method, make if-statements to catch any occurrence of the technique.
- Ensure that in each iteration any required value for the setter is being caught and stored in some variable
- Inside the if-statement, instantiate the created classes and set the desired values for the setter based on the caught values.

(c) Supporting new features in the GUI

- To add a new feature on the GUI, say a button, simply add the button on the controller class. Also, make a method which will be called when the button will be pushed.

- After that, open up the fxml file via SceneBuilder, and add the button. Make sure to adjust the size of the GUI in the properties section of SceneBuilder.
- After that, add the button, and put in the name of the button given in the controller class in the fx id. Similarly we can add a lot of features.