Design Document

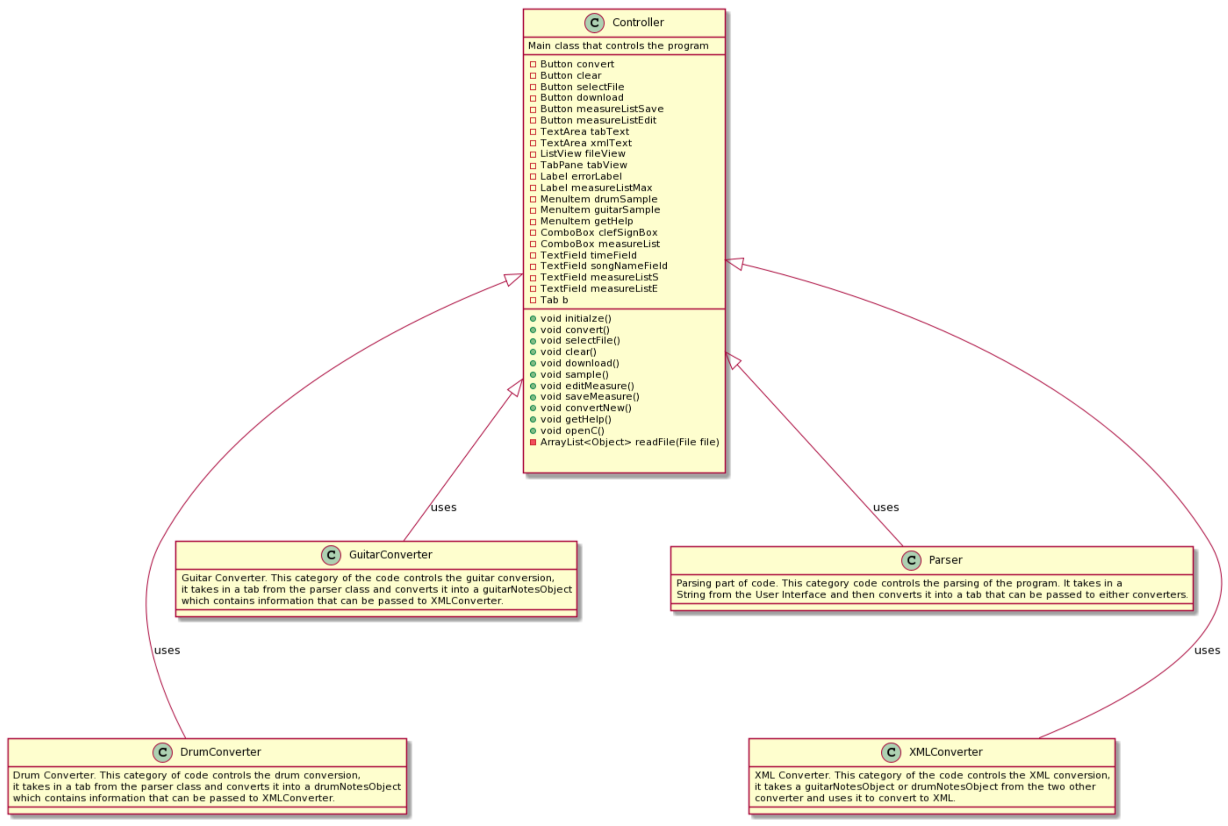
Tab2XML
Group 7

Joshua Genat
Andy Lin
Uthithmenon Ravitharan
Nicolae Semionov

Table of Contents

1.0 UML Overview

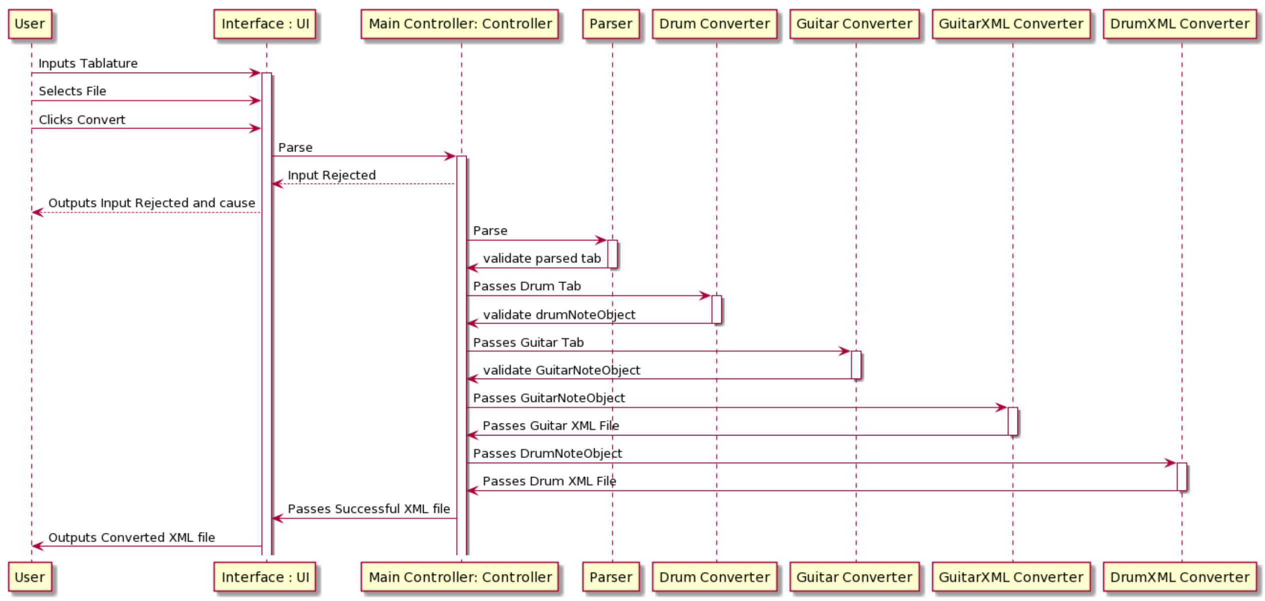
1.1 Class Diagram Overview:


Controller
Main class that controls the program
Button convert
Button clear
Button selectFile
Button download
Button measureListSave
Button measureListEdit
TextArea tabText
TextArea xmlText
ListView fileView
TabPane tabView
Label errorLabel
Label measureListMax
MenuItem drumSample
MenuItem guitarSample
MenuItem getHelp
ComboBox clefSignBox
ComboBox measureList
TextField timeField
TextField songNameField
TextField measureListS
TextField measureListE
Tab b
void initialze()
void convert()
void selectFile()
void clear()
void download()
void sample()
void editMeasure()
void saveMeasure()
void convertNew()
void getHelp()
void openC()
ArrayList<Object> readFile(File file)
uses
uses
uses
uses
GuitarConverter
Guitar Converter. This category of the code controls the guitar conversion, it takes in a tab from the parser class and converts it into a guitarNotesObject which contains information that can be passed to XMLConverter.
Parser
Parsing part of code. This category code controls the parsing of the program. It takes in a String from the User Interface and then converts it into a tab that can be passed to either converters.
DrumConverter
Drum Converter. This category of code controls the drum conversion, it takes in a tab from the parser class and converts it into a drumNotesObject which contains information that can be passed to XMLConverter.
XMLConverter
XML Converter. This category of the code controls the XML conversion, it takes a guitarNotesObject or drumNotesObject from the two other converter and uses it to convert to XML.

Controller.java

Controller class contains all the GUI components and the methods that are called when an action is done on these components, methods include convert(), which converts the pasted tab, selectfile() which allows the user to select a file that is a tablature to be converted, a clear() method which clears all fields, sample() method to get the sample tablatures, editMeasure(), saveMeasure(), and convertNew() for post conversion editing. getHelp and openC to open browsers for useful tools.

1.2 Sequence Diagram Overview:



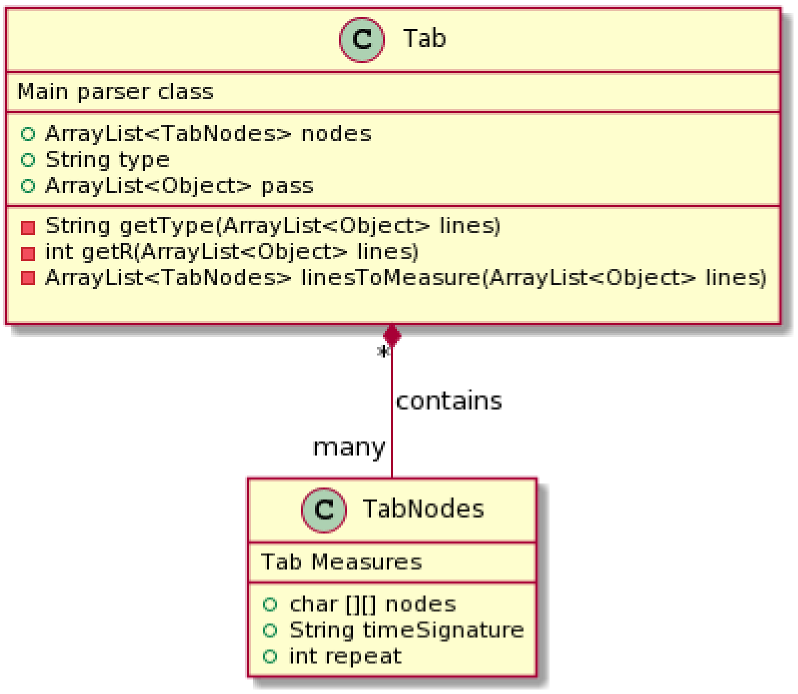
Steps.java

Controller.java is the main hub for the program, this is where all relevant information is passed from one class to another. The Controller contains multiple items and method thats control the GUI, the GUI then passes information to the controller, and the controller will then pass it to the parser. Parser will convert all of this to a Object Tab, and return it back to the Controller, the controller will then see if the type of tab is bass guitar, or drum, and will pass that accordingly to the converters. Finally the converters will pass back a NoteObject that contains all the information needed for the XML Converter to make the XML file

2.0 Parser UML

The Main Parser category. This category takes in the tab input from the user input, and converts it into an object that is readable for the converters.

2.1 Parser Class Diagram:


C Tab
Main parser class
○ ArrayList<TabNodes> nodes
○ String type
○ ArrayList<Object> pass
■ String getType(ArrayList<Object> lines)
■ int getR(ArrayList<Object> lines)
■ ArrayList<TabNodes> linesToMeasure(ArrayList<Object> lines)
*
contains
many
C TabNodes
Tab Measures
○ char [][] nodes
○ String timeSignature
○ int repeat

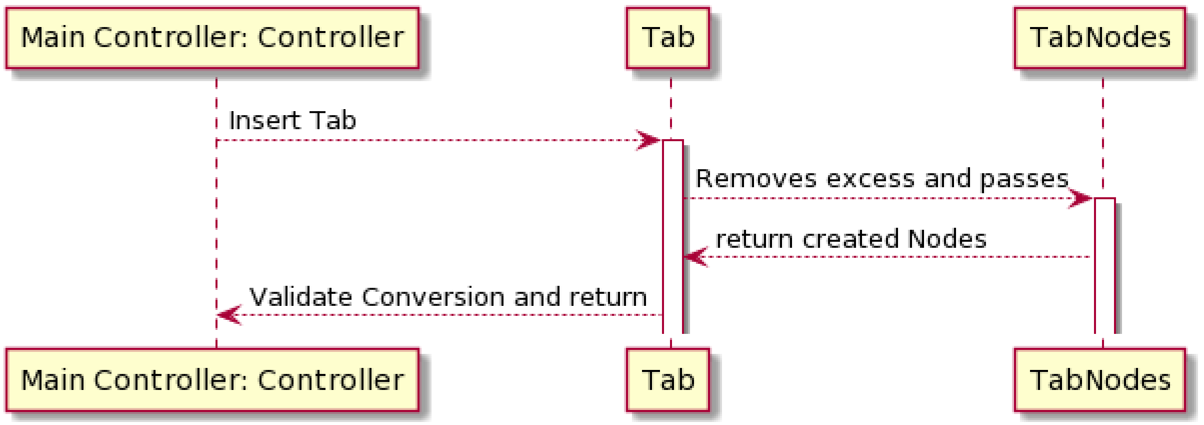
Tab.java

Tab.java is the main hub of the parsing, The tab class takes in a ArrayList<Object> which is the text that is to be parsed. It takes this input, gets rid of all excess lines that are not part of a tablature, and then it starts reading. It reads a line at a time, after getting a full line it will pass it onto a method called linesToMeasure, this will individually separate all the measures, assign it a repeat number through getR that will parse the repeats of the tablature, and also be assigned a time signature, this information would be passed to the tabnodes class to construct a tabnode(measure) that will then be added to the ArrayList of nodes. The main conversion process is dependent on what the Tab contains.

TabNodes.java

TabNodes.java is a information object class, it contains all the information needed in a measure that will then be passed onto the converters to convert it into notes and etc

2.2 Parser Sequence Diagram:

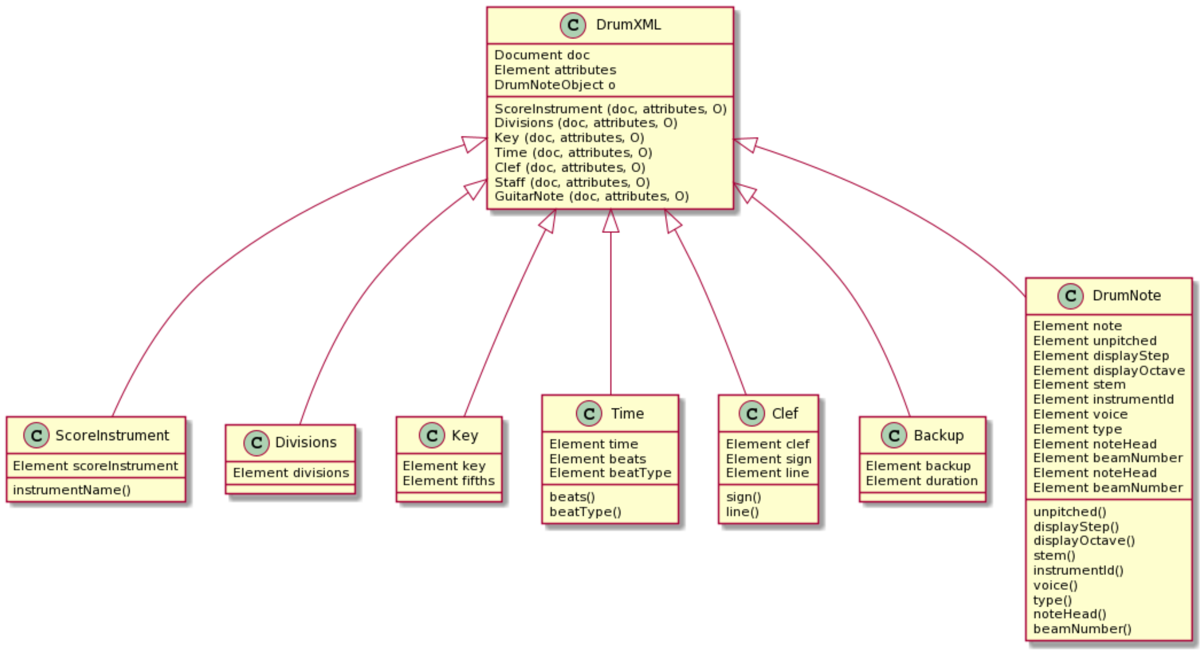


Main Controller: Controller
Tab
TabNodes
Insert Tab
Removes excess and passes
return created Nodes
Validate Conversion and return
Main Controller: Controller
Tab
TabNodes

3.0 XML Output

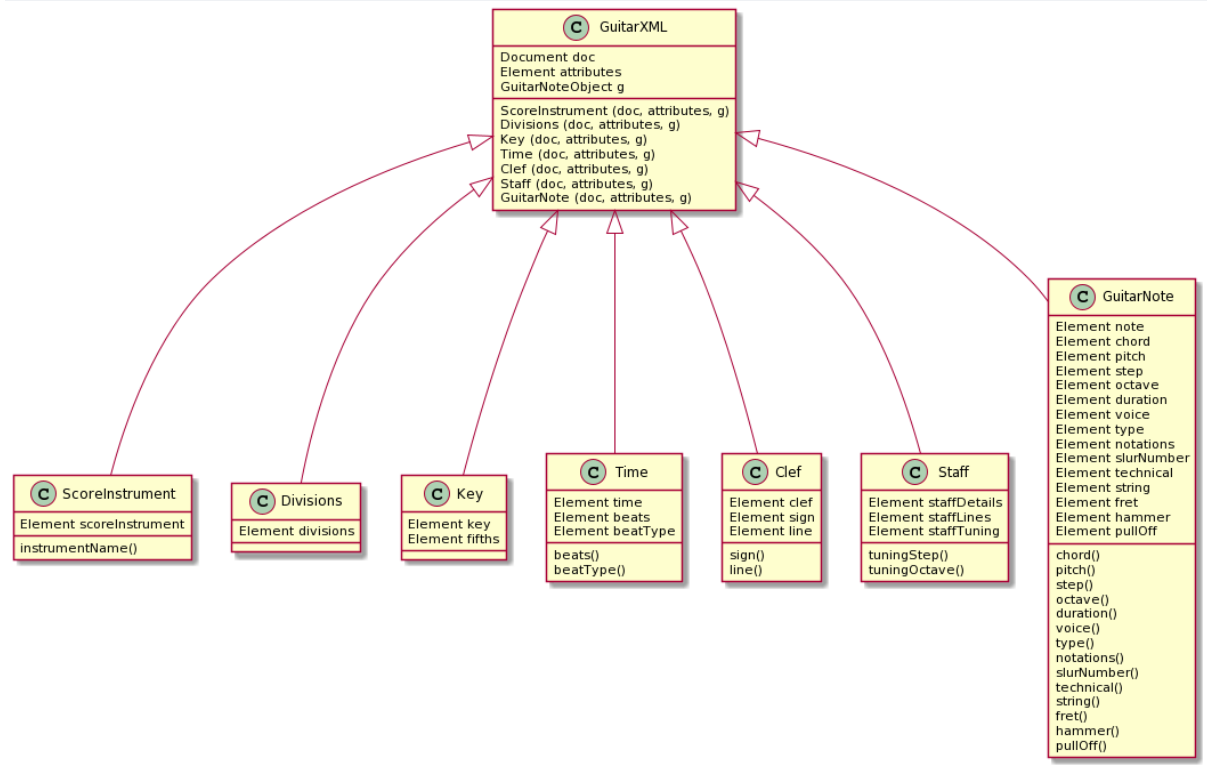
3.1 Drum XML

The Main DrumXML class initializes the XML Document, passing in the Document that will have the other attribute classes to append different sections, such as Divisions, Key, Clef, etc



3.2 Guitar XML

The Main GuitarXML class initializes the XML Document, passing in the Document that will have the other attribute classes to append different sections, such as Divisions, Key, Clef, etc



4.0 Converters

4.1 Guitar Conversion

The GuitarConverter class is the main component in the guitar conversion process. Its main function is to process a 2d array of characters and convert it into a form that's more accessible by the XML functions of our program. The converted output is in the form of an array of GuitarChord objects. GuitarChords are simply an array of notes with methods to manipulate multiple notes at once. GuitarNotes are objects that represent a single note, with the expected attributes such as string, step, fret, etc. This array is further passed down the program to be processed by the XML converters to create the final XML document.

GuitarConverter.java

GuitarConverter.java is the central component of guitar conversion within the program. It works by reading a 2d character array, column by column, and tries to locate and create notes. If the converter finds a ‘-’, that would be considered an empty space, and the converter creates a blank note. The converter only begins conversion once it reads the first blank, that would indicate the start of the tab. If the converter locates a number, that would be considered a valid note, and the note is added to a *GuitarChord* object, if other notes are found in the same column that would be considered a chord, and the boolean *isChord* for all other notes is set to true. The converter starts reading to the right to find other numbers or special characters. If there is a number on the immediate right of another number, that would be considered a two digit and are combined in the note generation (the second number will be skipped on the next column). If there is a ‘p’ or ‘h’ that would be considered a pull-off or hammer-on and the proper attributes are set in the *GuitarNotes* object of that note. If a ‘g’ is found, the number to the right will be considered a grace note, and the boolean *isGrace* is set to true.

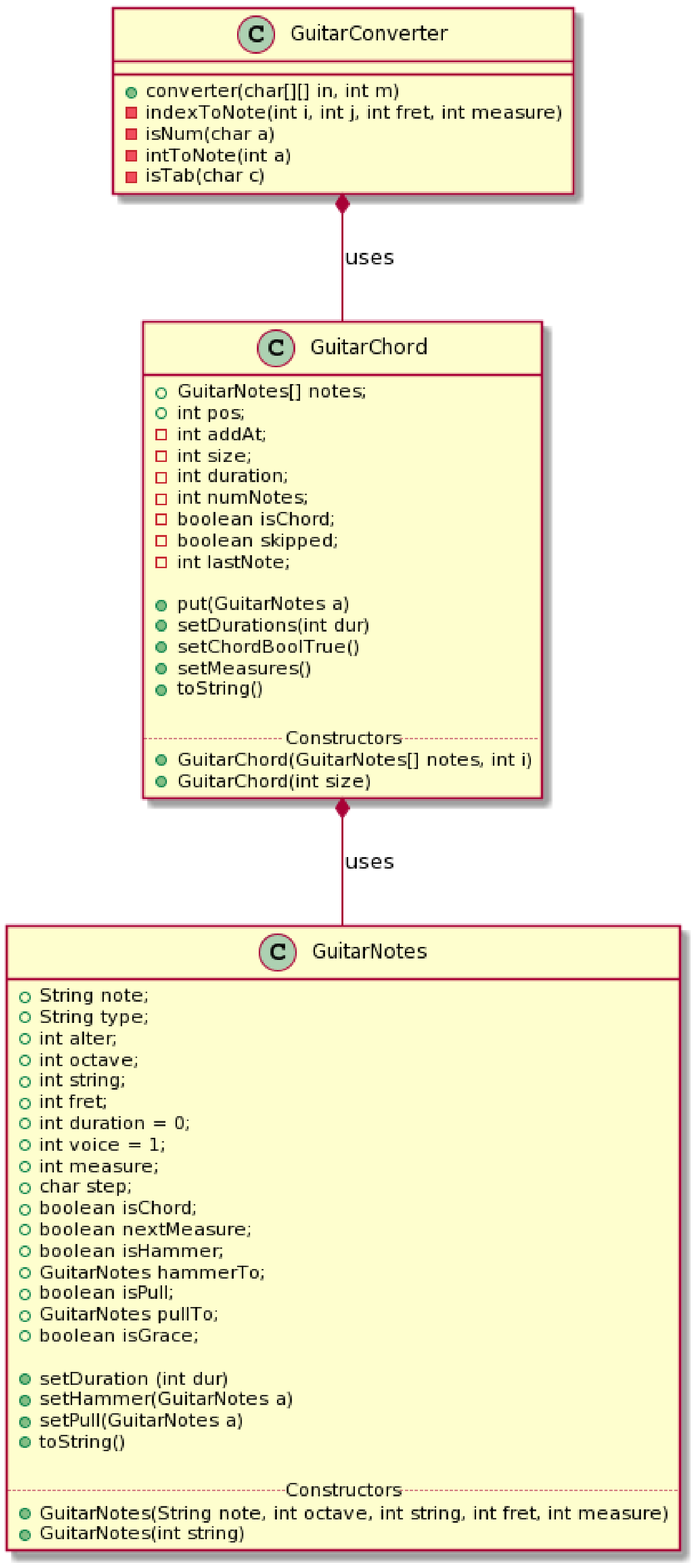
GuitarChord.java

The main purpose of GuitarChord is convenient storage of *GuitarNotes* objects. The converter creates an array of GuitarChords, each containing an array of GuitarNotes. The class has mutator methods that allows multiple notes to be edited at once. For example *setDurations(int)* sets all durations of the notes within the chord, without having to worry about other notes.

GuitarNotes.java

GuitarNotes are note objects that store all expected attributes needed for the later xml converters. These attributes are set as the converter moves along the column, in conjunction with *GuitarChord*. For example a new note will have its *isChord* boolean set to false, as the converter learns more about the tab, it may be set to true at a later stage if other notes are found on the same column.

4.1.1 Guitar Class UML



5.0 Drum Conversion

5.1: Drum Classes

DrumNoteObject.java

DrumNoteObject is the center class, which holds all the drum attributes obtained from the tablature. These attributes are all then passed on to the MusicXML converter to produce the desired MusicXML code. The DrumNoteObject class obtains these attributes from all the classes listed below. For Example, the DrumNoteObject class gets the total number of notes being played from DrumNoteCol class and gets the stem direction for these notes from the DrumStem class.

DrumRowSorter.java

The DrumRowSorter class finds out which drum instruments are being played, such as a Snare drum and a Hi-Hat. Using if statements, this class finds out the type of instrument from the two characters on the left side of most drum tablatures: example ‘CC’ equals Clash Cymbal. Knowing the type of instruments being played is essential for getting the voice number, and stem direction of a note.

DrumNoteRow.java

DrumNoteRow class contains a method called RowReader which is used to obtain the row values of the notes played in the drum tablature and puts them in their correct order. These row values are put into an arraylist and passed to the DrumNoteObject class. The row values arraylist, combined with the column value arraylist from DrumNoteCol, allows the MusicXML converter to see the locations of all the notes played, the order at which each note is played, and is essential for the calculation of the duration of each note later on in the program.

DrumNoteCol.java

DrumNoteCol class contains a method called ColReader which is used to obtain the column values of the notes played in the drum tablature and puts them in their correct order. Just like the DrumNoteRow class, these col values are put into an arraylist and passed to the DrumNoteObject class. The resulting arraylist combined with the arraylist formed in DrumNoteRow, allows the MusicXML converter to see the locations of all the notes played, the order at which each note is played, and is essential for the calculation of the duration of each note later on in the program.

DrumNoteHead.java

This class works very similarly to the DrumNoteCol and DrumNoteRow classes, except instead of returning the coordinates of the notes; the DrumNoteHead returns an arraylist of the character value of the technique used to play each note. For example, ‘o’ means strike and is usually used for drums, while ‘x’ means strike cymbal. This class is needed to identify whether an open or closed hi-Hat is being played.

DrumID.java

The DrumID class works with the DrumRowSorter, DrumNoteHead and DrumNoteRow classes to the part ID number from each note to the DrumNoteObject Class. The reason for the involvement of the arraylist from DrumNoteHead, is because one instrument (Hi-Hat) can be either opened or closed (both have different ID numbers) depending on the notehead.

DrumChordFinder.java

This class contains an if statement which is used to identify if a note needs to have the “<chord> tag.

DrumVoice.java

This class uses the information obtained from the RowSorter and DrumNoteRow classes to assign a voice value for every note.

DrumDisplayOctave.java

This class uses the information obtained from the RowSorter class (which instrument is being played) to assign a display octave value to every note.

DrumDisplaySteps.java

This class uses the information obtained from the RowSorter class (which instrument is being played) to assign a display step value to every note.

DrumStem.java

DrumStem finds the stem direction of each note using the Voice values obtained from the DrumVoice.java class.

DrumDuration.java

DrumDuration uses the arraylist obtained from DrumNoteCol, mixed with the information obtained from the RowReader and ChordFinder classes to determine how long each note is played for. The duration is important for identifying the type of note being played and for the formation of beams.

BackUpFinder.java

BackUpFinder uses the voice values obtained from DrumVoice to return a boolean arraylist, which tells the XML converter when to include the <Backup> tag for the MusicXML code.

DrumFlam.java

The DrumFlam class looks through the tablature, and finds out if any notes are a flam (grace notes for drums). This is used to tell the XML to print the flam note twice with a slur attached to both notes.

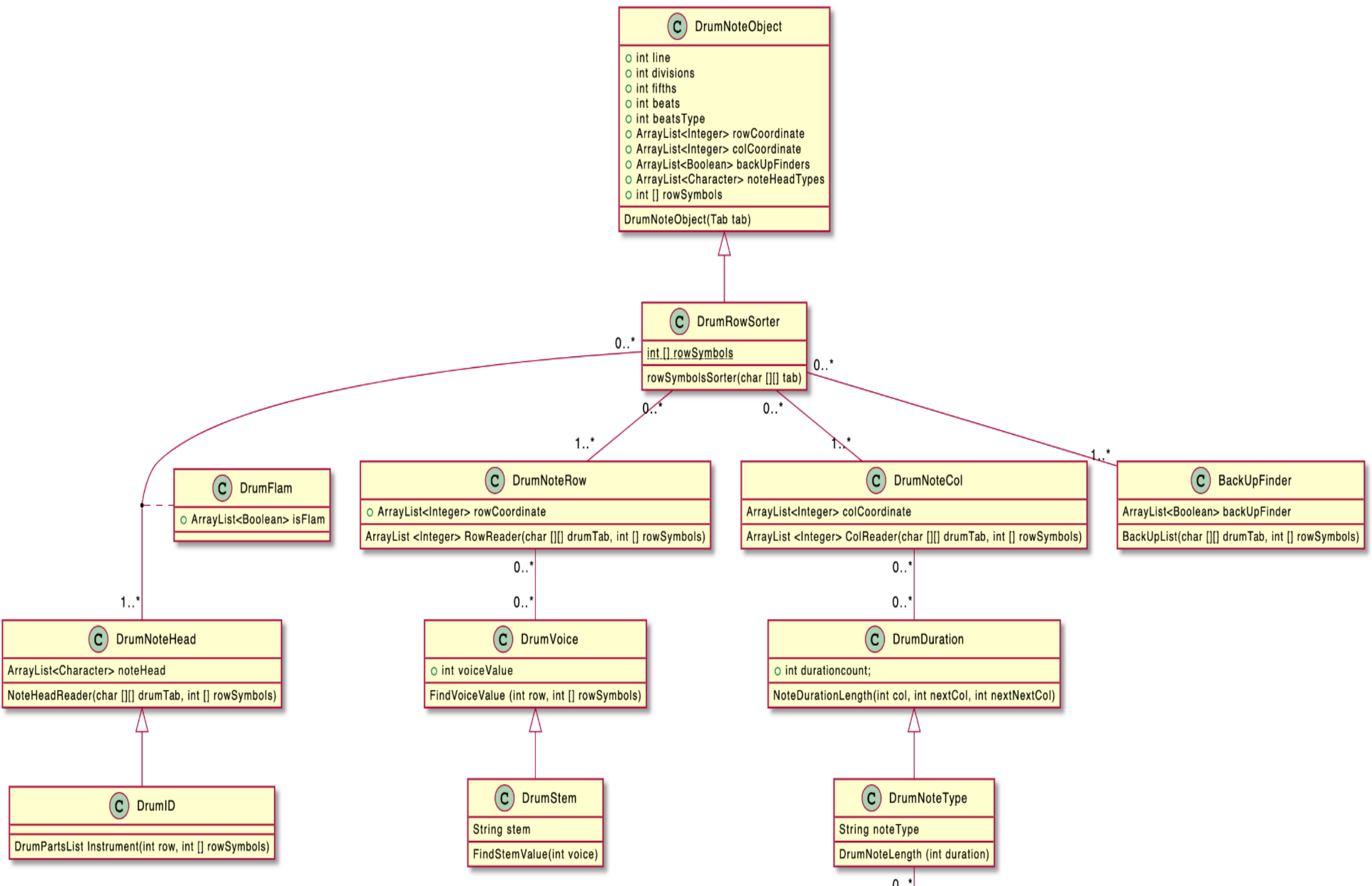
DrumBeamNumber.java

The DrumBeamNumber class is used to determine whether a note is part of a beam. If the note is part of the beam, the class then determines the notes role inside the beam such as “start”, “continue” and “end”. This class contains two methods, the first method “BeamOneStatus”, determines the status of the first beam number, or null. The second method “BeamTwoStatus” does the for beam number 2.

DrumNoteType

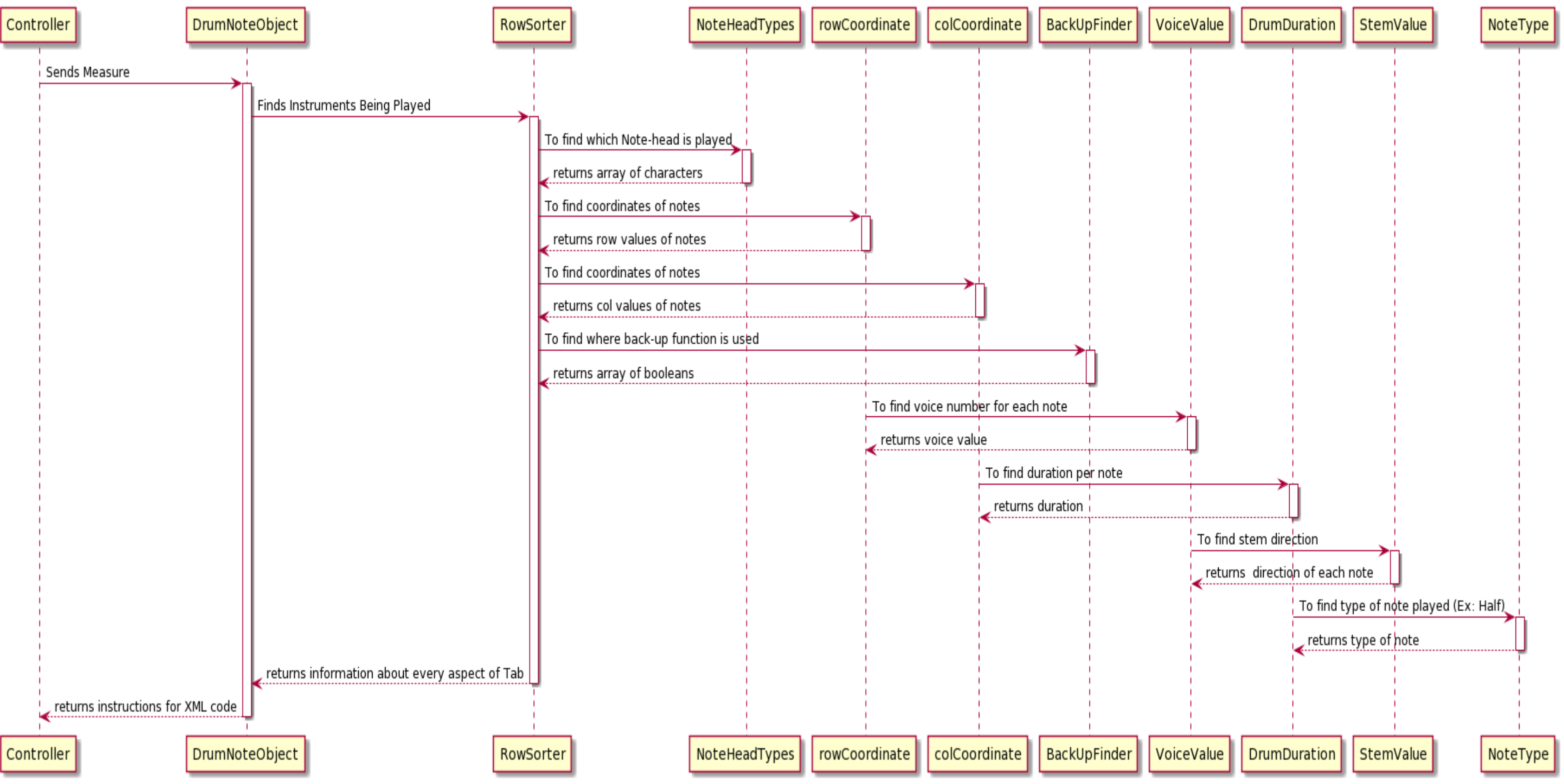
Using the duration and the division of the tablature (User input), this class determines the type of note being played, such as an eighth or half note

5.2 Drum Class Diagram



DrumNoteObject
int line
int divisions
int fifths
int beats
int beatsType
ArrayList<Integer> rowCoordinate
ArrayList<Integer> colCoordinate
ArrayList<Boolean> backUpFinders
ArrayList<Character> noteHeadTypes
int [] rowSymbols
DrumNoteObject(Tab tab)
DrumRowSorter
int [] rowSymbols
rowSymbolsSorter(char [][] tab)
0..*
0..*
0..*
0..*
1..*
1..*
1..*
1..*
DrumFlam
ArrayList<Boolean> isFlam
DrumNoteRow
ArrayList<Integer> rowCoordinate
ArrayList <Integer> RowReader(char [][] drumTab, int [] rowSymbols)
DrumNoteCol
ArrayList<Integer> colCoordinate
ArrayList <Integer> ColReader(char [][] drumTab, int [] rowSymbols)
BackUpFinder
ArrayList<Boolean> backUpFinder
BackUpList(char [][] drumTab, int [] rowSymbols)
0..*
0..*
0..*
0..*
DrumNoteHead
ArrayList<Character> noteHead
NoteHeadReader(char [][] drumTab, int [] rowSymbols)
DrumVoice
int voiceValue
FindVoiceValue (int row, int [] rowSymbols)
DrumDuration
int durationcount;
NoteDurationLength(int col, int nextCol, int nextNextCol)
DrumID
DrumPartsList Instrument(int row, int [] rowSymbols)
DrumStem
String stem
FindStemValue(int voice)
DrumNoteType
String noteType
DrumNoteLength (int duration)

5.3 Drum Sequence Diagram


Controller
DrumNoteObject
RowSorter
NoteHeadTypes
rowCoordinate
colCoordinate
BackUpFinder
VoiceValue
DrumDuration
StemValue
NoteType
Sends Measure
Finds Instruments Being Played
To find which Note-head is played
returns array of characters
To find coordinates of notes
returns row values of notes
To find coordinates of notes
returns col values of notes
To find where back-up function is used
returns array of booleans
To find voice number for each note
returns voice value
To find duration per note
returns duration
To find stem direction
returns direction of each note
To find type of note played (Ex: Half)
returns type of note
returns information about every aspect of Tab
returns instructions for XML code

6.0 Maintenance Scenarios:

Addition of Instruments: Adding instruments in the future is easy but not completely optimized, as the developer would first need to create a tab to instrument converter. But then because our XML converters are not generalized for all instruments currently, the developer would also need to create a similar XML converter but with any extra functionality that the new instrument requires. For example the drum had backup but the guitar did not. Other than that the user would then just have to add into the UI and controller the new options and everything else should work accordingly.

Addition of time signature directly from tab: If a developer would like to add a feature where the user can just enter the time signature over a measure they can edit this straight from the Tab and TabNodes classes. The developer would have to create methods to recognize the timesigantures and then pass this onto the tabnodes. The developers would then have to set the time signatures of the tab nodes to the corresponding input from the user.

XML Document: When adding a new section to the XML Document, the developer would need to first locate which instrument this belongs to, and this can be guitar or drums or both. Based on where you will find this section, the developer would need to append it accordingly.
For example, if “pull-offs” were to be added, first locate the Guitar class and determine that pull-offs should be located in the “note” section of the XML, inside the “technical” section. Then append it to the document.