Design Document

TAB2MXL: Text File to MusicXML File Converter

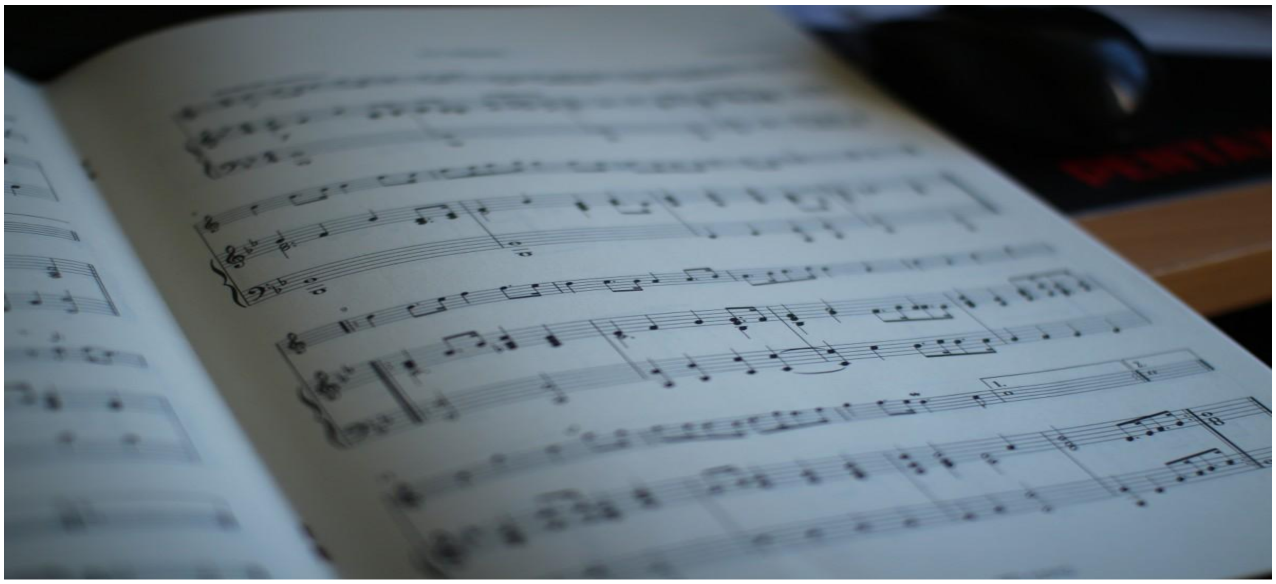


Table of Contents

1.0 Overview

TAB2MXL is a desktop application designed to convert text tablature files into their equivalent MusicXML format files to be used in external music editing software, such as MuseScore. The software supports conversion of text tablatures for guitar, drums and bass. The conversion software features an intuitive graphical user interface that is compatible with most operating systems. The translated files will appear in the interface after successful conversion and can be saved to the user's device.

1.1 Document Scope and Purpose

This document will describe the technical design of the TAB2MXL software. The document's purpose is to describe the architectural overview of the software and dissect the overall structure into its most essential components. Each component will be discussed in detail, including its methods and classes and how it contributes to the project scope, system architecture, design specification.

UML diagrams for both the front end and back end are provided as a visual aid to enhance understanding of how individual components function in relation to each other. The UML diagrams will include classes, as well as the attributes and methods that belong to each class to further deepen understanding of how the various classes operate and contribute to the overall software architecture.

This document functions as a foundational reference point for developers. At the end of this document, a section is dedicated to backend maintenance, written with the intention to guide future TAB2MXL developers in the implementation of new features. Please note, this is a baseline document and will be updated as the development progresses.

1.2 Target Audience

This document is targeted towards the following technical stakeholders:

- Development Team
- IT Management
- Supporting Staff

1.3 System Environment

- Integrated Development Environment: Eclipse 2020-09 (4.17.0)

- JavaSE - 1.8

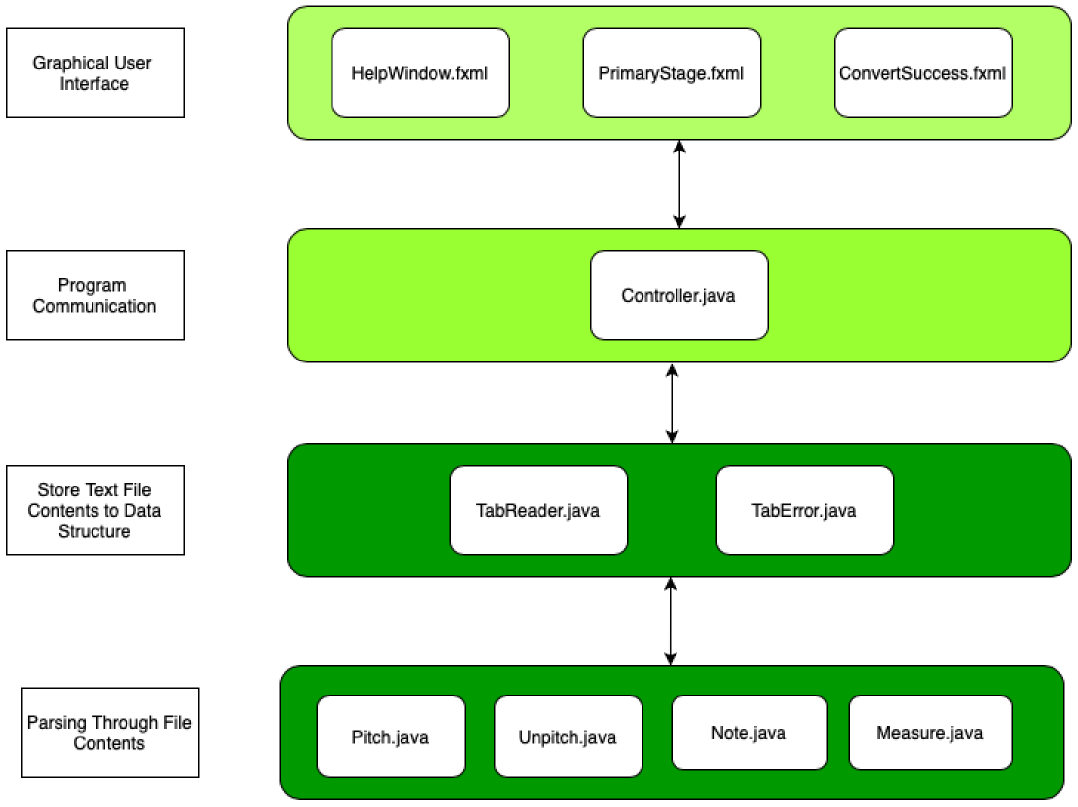
TAB2MXL has a number of external dependencies, which are handled by Gradle (Version 6.3) including:

- Jacoco (Version 0.8.6)
- Junit (Version 4.12)
- JavaFX (Version 0.0.8)

1.4 Project Scope and Limitation

Our software system takes in an input text file containing instrument tablature and converts it to its equivalent MusicXML file. The project is limited to input text tablature for the following instruments: guitar, bass and drums. More specifically, bass guitar text tablature is limited to 4-string bass guitars. Guitar tablature with 5 or more strings will be interpreted as a classical guitar. Drums text tablature is limited to the conventional 9-piece drum set. Inclusion of additional components of the drum set, such as china cymbals, will trigger an error when the user attempts conversion. Uploading text tablature for instruments beyond the ones mentioned in this section will result in unsuccessful conversion.

2.0 System Overview



The TAB2XML software can be broken down to four major sections:

1. Graphical User Interface (GUI)

The graphical user interface includes all components that the user can use to communicate with the system. Back end processes are initiated via user interaction with the GUI. In our design, a resource folder contains a number of .fxml responsible for generating the visual appearance of the GUI. Different .fxml files are called and loaded to appear in the GUI based on user interaction. The selection and calling of these methods happens in the next stage. This interface will display the final output of the file conversion.

2. Program Communication

The Controller class is critical to the system's flexibility, as it allows the user to make modifications to both the input text file and output MusicXML file. This class is also responsible for error-handling and communicating the nature of errors to the user. The Controller class receives the user decisions, sends the user file to the back end and returns the resultant MusicXML file to the GUI.

3. Import, Filter and Parse Text File Contents

At this stage, the text file has been uploaded by the user and the text file is read into the back end. The text is organized into a linear array data structure. The text is interpreted to determine the instrument type of the input tablature. The text array is scanned and filtered to remove unnecessary or unrecognized notation. This component of the software sends the completed output MusicXML file back to the controller in the previous component.

4. Create Object Representation of MusicXML Elements

The filtered content is parsed and the parsed data is used to construct objects of multiple classes. The types of objects constructed from the parsed data vary depending on the instrument detected in the previous stage. These objects are compiled and organized in a hierarchical manner and are ultimately used to translate the input data to MusicXML file format.

3.0 Front End Design

The front end software design relies primarily on the JavaFX Software Development Kit (SDK), which is imported into the project as a Gradle plugin. The user interface includes a button to upload a text file, a drag and drop file upload feature, and a copy and paste input text feature. Once the input text has been uploaded, the GUI enables buttons to modify elements that will appear on the output MusicXML file, including the title, composer, time signature and key. Additional buttons on the GUI allow the user to enter a range of measures to edit in a text window either before or after conversion or enter a range of measures for which they would like to assign a unique time signature. After the convert button is clicked, the GUI informs the user of successful conversion or errors events by producing pop up windows containing the appropriate text for the situation.

3.1 Front End Classes

Within the TAB2XML project, there is an application package containing three classes: Controller, Main and MainLaunch. These three classes make use of packages available in the JavaFX SDK. More specifically, TAB2MXL front end software imports the javafx.application, javafx.stage, javafx.fxml, javafx.scene.controller, and javafx.event packages.

3.1.1 Main Class

The Main class extends the Application class from the javafx.application package. The Main class imports the FXMLLoader class from the javafx.fxml package to call the static method load(URL location). This method is required to access the .fxml files located in the application package in the TAB2MXL project resources folder. These .fxml files are written in xml-based language and contain code to produce the visual components of the GUI, such as the panes, buttons, labels, colors, fonts, and the position of these components in relation to each other. The Main class initially loads the PrimaryStage.fxml file, which generates the TAB2MXL home screen.

Important Methods in Main Class:

- `load(URL location) - Loads an object hierarchy from a FXML document`

3.1.2 MainLaunch Class

The MainLaunch class imports the Application class. The Application class contains the static method launch(Class<? Extends Application> appClass, String arg), which is required to launch a standalone application implemented using the JavaFX SDK 15. The method is called with the Main class as an input parameter.

Important Methods in MainLaunch Class:

- `launch(Class<? Extends Application> appClass, String arg) - Launches a standalone application`

3.1.3 Controller Class

The GUI reacts based on how the user interacts with the software, and subsequently, on which methods are called in the Controller class as a result of these interactions. The Controller class imports a number of classes from the javafx.scene.control, javafx.stage, javafx.event packages, including the Button, TextField, RadioMenuButton, ActionEvent and Window classes. These classes work together to generate the appearance of graphical components (Button) and their associated reactions to user interaction (ActionEvent). For the sake of brevity, these classes and their methods will not be discussed in further detail. Further information regarding the functionality of these classes can be found in *The Java Tutorials*, available at https://docs.oracle.com/javase/tutorial/. More importantly, the Controller class instantiates an object of the TabReader class, which is the back end class responsible for a significant portion of the text to MusicXML conversion procedure. The TabReader object is used to call methods from the Controller class that connect the front end and back end of the TAB2MXL software. These methods allow the Controller class to send text input to the back end and display the resultant MusicXML output file on the GUI. It is also used to call methods that send user-selected or user-entered data from the front end to the TabReader class, such as saveKey() and saveTitle(), so it can be reflected in the output MusicXML file.

Important Methods for Controller Class:

- `selectKey()- Allows the user to select the different key majors.`
- `selectComposer()- Allows the user to enter composer name of the tablature/song.`
- `selectTitle() - Allows the user to enter the title name of the XML file.`
- `selectTimeSig()- Allows the user to select different time signatures.`
- `helpClick()- Will guide the users on how to operate the system.`
- `startClick()- Responsible for the users to start the program.`
- `convertClicked()- Allows the user to convert the tablature to the MusicXML file`

3.2 Front End Maintenance

In order to add new features on the GUI there are a few methods that can be added into existing classes including: Controller.java class located under src/main/java and in the package called application. The technique used to design the features on the GUI includes writing .fxml files and loading all the elements from the FXML file to the controller using the tag "@FXML".

An example of this is when the help button was implemented. Under TAB2XML, you can locate a file named src/main/resources , this is where all the .fxml files are located and will be called from in the controller.java class. The implementation of the .fxml file for the help button sets the font name and size, it also sets the text included when the help button is pressed. It includes the different features within the button such as height, width, border color and so on.

In the controller.java class a method is made starting with “@FXML” with the method written underneath called the .fxml file. Controller class includes the logic and function of the elements.

Furthermore, in order to add a new feature on the GUI. Start off by writing the .fxml file for the designated feature you want to create by importing the different javafx.scene.control. The different imports include changes to the layout, label, text area and the font.

For the future, various new elements can be added to the front end. To start, to see the input files text and the translated text in a larger size, one can add a ZoomEvent to any of the specified TextArea the controller class. Other changes in the front end include having the option for the user to have their recently translated files on a menu bar on the side. This would include updates to the primary stage to include a menu bar and we would need to call the Preference class as it would help list recently opened items in the menu bar. Furthermore, this feature would demand a dynamic build. To achieve dynamic build, we could implement an Action that extends the AbstractionClass to deal with when the user attempts to open a file. Next, when the Action is successful running, it would require the newly accessed file's name to the Preferences class and dynamically rebuild the menu bar.

In the front end designing is greatly dependent on the features that the backend can support. Front end designing for this project was limited to Controller and the usage of fxml files.

4.0 Back End Design

The back end software processes the input text file in two stages. First, it reads the input file and inserts it into a linear array data structure. This array is then used to detect the instrument type. Once the instrument has been determined, the array is scanned to remove notation that is not recognized by TAB2MXL for that instrument’s tablature. For example, if the instrument is a guitar, the filtration process will remove occurrences of the letter “o”, whereas if the instrument is drums, the filtration process will remove all occurrences of numerical digits. The second stage consists of using the filtered and parsed data from the first stage to instantiate objects of various other classes. The object instantiated will depend on the character values parse from the text, as well as the instrument detected in the first stage.

4.1 Back End Classes

The back end classes rely on importing two Java packages into the TAB2MXL package within the TAB2MXL project. The java.io package is imported to instantiate the File object provided by the user. The java.util package is imported to provide access to a number of classes that implement linear, map and set-based data structures that can be instantiated to aid in the organization of the data parsed from the input file.

4.1.1 TabReader Class

An object of the TabReader class is instantiated in the Controller class when the user uploads a text file. When the user clicks convert on the GUI, TabReader class methods are called to perform the necessary steps for conversion. When a TabReader object is created, it instantiates a number of ArrayList objects, which are used to organize the parsed data in a hierarchical manner, since MusicXML files are written in an xml-based hierarchical structure. These ArrayLists include: guitarTuning, measureElements, allMeasures, scoreInstrument, techniques and drumsetTechniques. The ArrayList measureElements is a list containing elements of data type Measure and is used to organize the measures, which are in turn, used to organize notes. The remaining lists are lists of Character and String data types, and are used to identify and interpret data from the input text file such as which characters are considered notes or special musical techniques and which characters should be ignored altogether. This is done by comparing characters from the input text to the ArrayLists techniques and drumsetTechniques, which contain a fixed set of characters that represent all of the currently implemented features of guitar and drum instruments in TAB2MXL, respectively. Characters not found in the two previously mentioned ArrayLists are ignored. The following methods are called sequentially to extract this information: setInput(File inputFile), readFile(File inputFile), filterInput(), getInstrument(), convertTabs(), makeNotes() (or makeDrumNotes()), sortArray(), setDuration(), noteDuration(), noteType() and finally toMXL(). Below are brief descriptions for each of the methods list.

Important Methods for TabReader class:

- `setInput(File inputFile)- Splits every line in the text file and puts it into an array.`
- `readFile(File inputFile)- Reads the text file in order to convert an XML file/`
- `filterInput()- Detects unsupported characters and removes them.`
- `getInstrument()- Detects which instrument (guitar, bass or drum) is in the tablature.`
- `convertTabs()- In charge of mainly error handling. Tells the user where the error is.`
- `makeNotes() (or makeDrumNotes())`
- `sortArray()- Sorts the measures in ascending order for the XML file.`
- `setDuration()- Sets the calculated duration to the notes.`
- `noteDuration()- Calculates the duration of the different notes in the tablature.`
- `noteType()- Based on the duration of the note, the duration of the note is set to a specific note type.`

- `toMXL() - A string method that will contain the designated XML file.`

4.1.2 TabError Class

The TabError class works with the TabReader class and the Controller class to identify and alert the user of issues with the input text file. The convertTab() method in TabReader returns a TabError object or it returns null, depending on whether the compileMeasures() and makeNotes() methods were able to complete successfully. If either of these methods run into an issue, the TabError object is instantiated with the appropriate error message, the measure number where the error occurred, as well as a String representation of the measure itself. This class contains getter methods that allow us to retrieve this information and return it to the user in the GUI.

Important Methods for TabError Class:

- `getMeasureNumber()- Returns an integer value at the measure the error occurs.`
- `getMeasure()- A string containing a set of a measure where the error is obtained.`
- `getErrorMsg()- A string containing an error message.`

4.1.3 Attributes Class

The Attributes class has one primary function. It sets the time signature (attributes beat and beatType) and the key value for the output MusicXML file using user-selected data or preset default values. The toString() method of the Attributes class is overridden to first check the instrument value of the TabReader object, so that it can concatenate the appropriate attributes to the final output MusicXML String.

Important Methods of Attributes Class:

- `setKey()- Sets the different key majors.`
- `toString()- A string the contains the XML file for attributes only.`

4.1.4 Measure Class

When an object of class Measure is created, it instantiates an ArrayList containing elements of data type Note. The primary function of this class is to act as a container for Note objects. Once the final Note object of a Measure has been stored in this ArrayList, the Measure is added to an ArrayList in TabReader, which collects all completed Measures. The Attributes element falls under the Measure element in the MusicXML hierarchical format, as such, Measure contains an instance variable of the Attributes class, so the Attributes toString() method is called within the Measure toString() method. This ensures that all elements appear in the correct order in the output MusicXML file. The Measure class contains methods that aid in determining duration and type values for Note objects. The method getIndexTotal() checks the length of the String representation of the measure. This integer value is used in conjunction with the time signature and the difference between Note index values to approximate the duration value of the given Note. This is done using the getNotes() method, which returns an ArrayList of Notes of the Measure. The ArrayList is used to compare each Note's character index value with the Note that precedes it in the ArrayList.

Important Methods of Measure Class:

- `sortArray()-Sorts the measures in ascending order.`
- `getNotes()- Gets the notes of the tablature.`
- `getIndexTotal()- Returns the length of the string representation of the measure.`

4.1.5 Note Class

The Note class contains a long list of boolean instance variables, which are all initially set to false. These instance variables represent the features recognized by TAB2MXL for guitar, bass and drums. When the makeNotes() or makeDrumNotes() method is called in TabReader, multiple Note objects are created. The Note constructor called depends on the instrument type. Each boolean variable is associated with a specific notation, if the Note object is instantiated with a character value that is associated with one of these boolean variables, that particular variable assignment value is changed to true. The Note toString() method checks which of these variables are set to true, and includes them in the MusicXML String representation of the note.

Important Methods of Note Class:

- `modifyNote(String note, String instrument)`
- `drumsetNoteHead()- Returns a mapped list that contains the different drum note heads.`
- `toString()-A string containing the XML output of notes.`

4.1.6 Pitch and Unpitch Classes

The Pitch and Unpitch class serve a similar purpose. The Pitch class constructor is called in the Note constructor if the instrument type is guitar or bass. This constructor uses the string number, fret number and string tuning value to determine the step, octave and alter value for the Note that called it. The Unpitch class constructor is called in the Note constructor if the instrument type is drums. This constructor uses the instrument abbreviation at the beginning of the measure and note character with preset Maps to determine the step, octave and instrument ID for the Note that called it.

Important Methods of Pitch Class:

- `initGuitarOctaves()- Returns a mapped list that contains the different octaves for guitar.`
- `initAllNotesMap()- Returns a mapped list that contains the different keys.`

Important Methods of Unpitch Class:

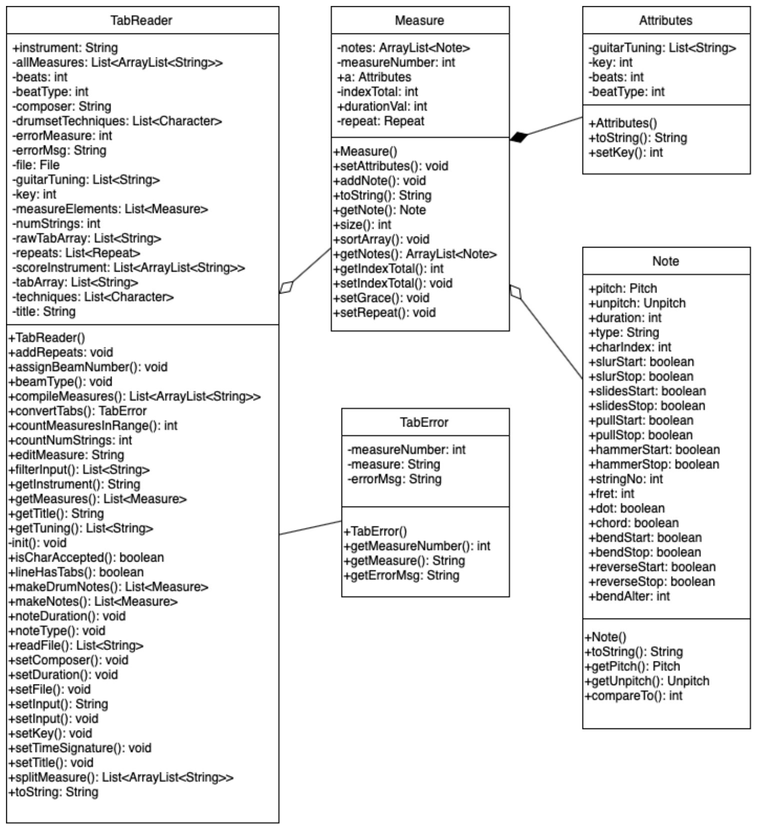
- `initDrumsetSteps()-Returns a mapped list that contains the different drum steps.`
- `initDrumsetOctaves()-Returns a mapped list that contains the different drum octaves.`
- `initDrumsetIDS()-Returns a mapped list that contains the different drum instruments.`

4.2 Back End Maintenance

To add a feature such as support for a new guitar technique, one will need to go to the TabReader class and add the character of the new technique to the techniques list. To do this go to the init() method and change techniques.addAll(Arrays.asList('\\', '/', 'b', 'g', 'h', 'p', 'r', 'S', 's')); to techniques.addAll(Arrays.asList('\\', '/', 'b', 'g', 'h', 'p', 'r', 'S', 's', '<NEW CHAR HERE>'));
Next do ctrl + f and paste the following: if (techniques.contains(currentLine.charAt(k)))
Go into this if-statement and add another if-statement:
if (currentLine.charAt(k) == '<NEW CHAR HERE>') {...}
Now inside this if-statement, one will have to refer to the MusicXML documentation and the rest of this documentation in order to complete the implementation of this feature.

5.0 Back End UML Diagrams

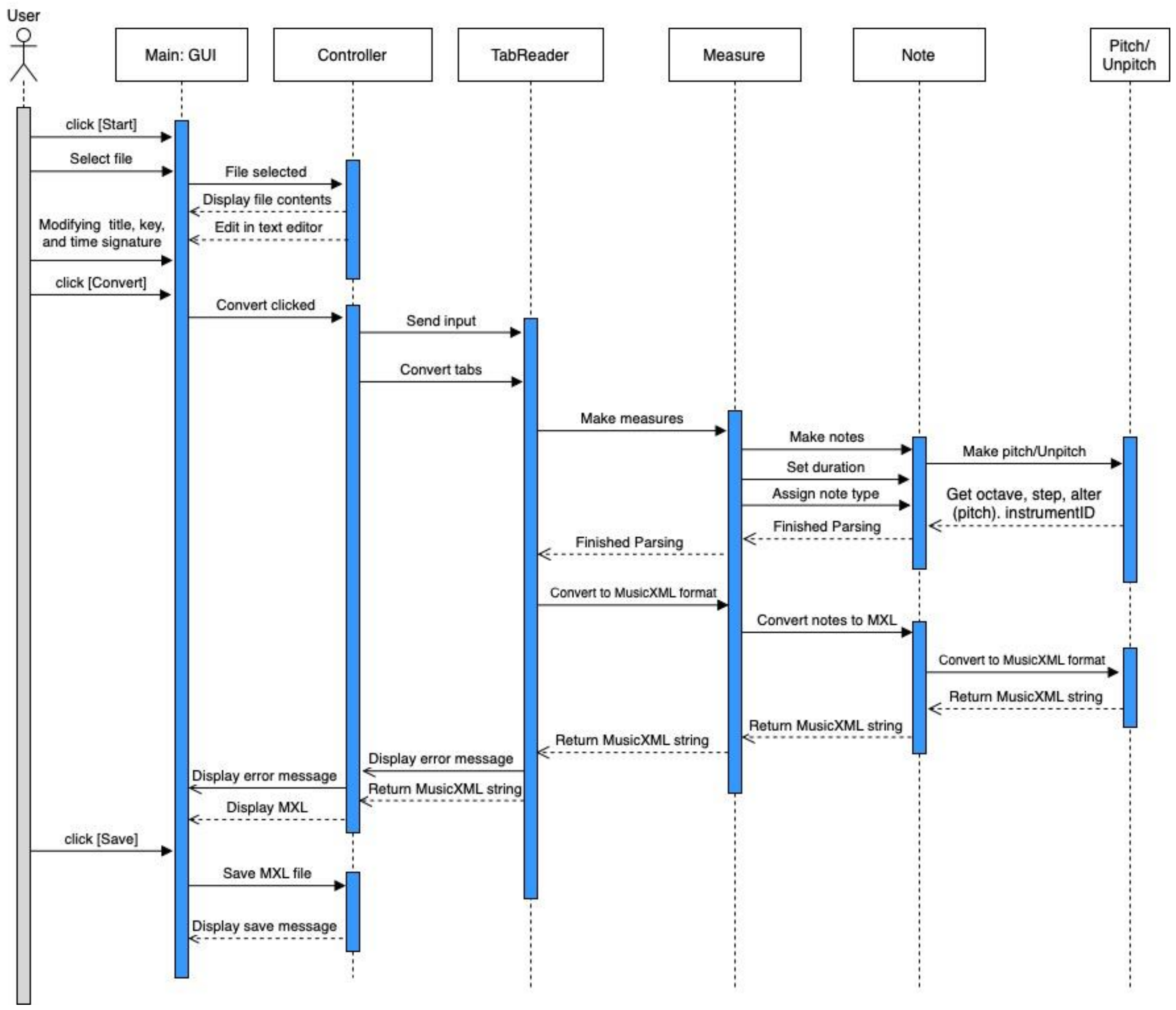
5.1 Back End Class Diagram



5.1.1 Interaction and Connection to Each Other

The interactions between the back end classes are shown in the class diagram above. The class Attributes and Note classes interact with the Measure class and the Measure and TabError class interact directly with the TabReader class. TabReader contains an ArrayList of Measure objects and in turn, Measure contains an ArrayList of Note objects. This organization allows Measure objects and Note objects to be formed sequentially to reflect the order in which they appear in the text tablature. TabError is directly connected to TabReader; the TabReader method convertTab() returns a TabError object if issues are encountered during the conversion process. All classes shown in the class diagram are found in the TAB2MXL package in the src/main/java folder, as such, access modifiers have been assigned so that methods for one class may be called from another class.

5.2 Back End Sequence Diagram

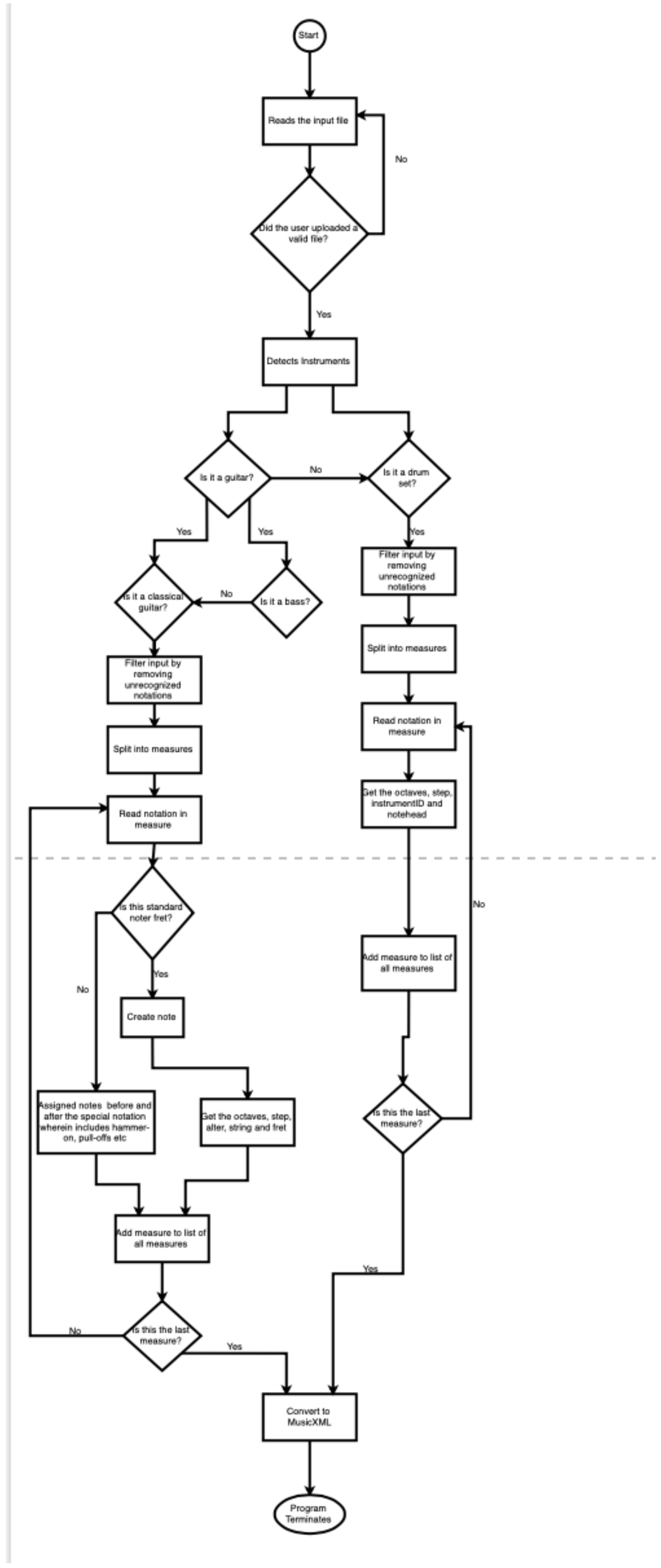


5.2.1 Interaction and Connection to Each Other

The sequence diagram shows how the classes interact with each other and what happens to each of the classes before proceeding to the next. The interaction between the classes is that methods written in classes of the back end need to connect to the front end. This is important to demonstrate how the connection between the two ends. The GUI performs tasks such as uploading a text file or displaying the contents of the successfully converted output file.

To analyze the interaction and connection between the classes in the sequence diagram, the Main class is run through Gradle and is responsible for the GUI. TabReader is used to parse through the text file getting information such as the duration, the note types and so on. It is then returned to build the MusicXML file. The measure class is created to get the elements and content of the measure. The variables created in measure are then called into the TabReader. The measure class parses the tablature and returns the MusicXML format of the measure. The note class is then used to identify the different notes and the grace notes, it is then returned in MusicXML format. The class pitch returns the data of the pitch class which are the octaves, steps, and alters of the tablature uploaded. Once it obtains these information, as one can see from looking at the diagram it starts building the MusicXML file. At the final stage of the application in main it returns the designated MusicXML file.

5.3 Back End Activity Diagram

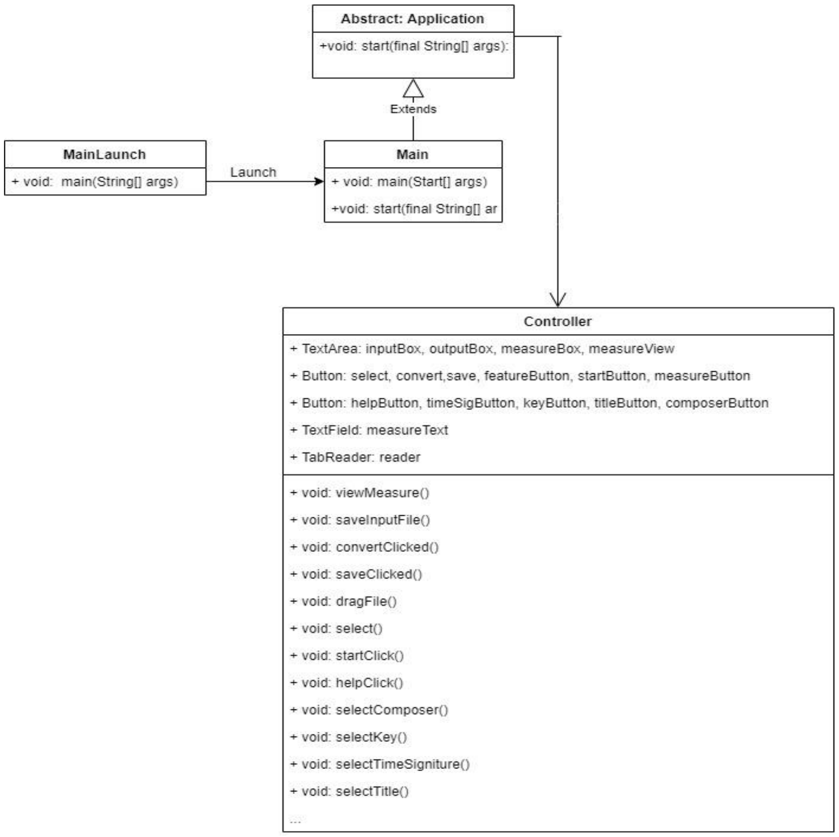

Start
Reads the input file
No
Did the user uploaded a valid file?
Yes
Detects Instruments
Is it a guitar?
No
Is it a drum set?
Yes
Yes
Yes
Is it a classical guitar?
No
Is it a bass?
Filter input by removing unrecognized notations
Split into measures
Read notation in measure
Is this standard noter fret?
Yes
No
Create note
Assigned notes before and after the special notation wherein includes hammer-on, pull-offs etc
Get the octaves, step, alter, string and fret
Add measure to list of all measures
Is this the last measure?
No
Yes
Filter input by removing unrecognized notations
Split into measures
Read notation in measure
Get the octaves, step, instrumentID and notehead
No
Add measure to list of all measures
Is this the last measure?
Yes
Convert to MusicXML
Program Terminates

5.3.1 Interaction and Connection to Each Other

The activity diagram demonstrates how the three instruments in this project are determined and how the input text is filtered and parsed. In the GUI the user will upload or drag a text file that contains a tablature of the valid instruments. The software system will determine whether it is a drum set or guitar/bass, if neither is detected an error is returned. The instrument determined at this stage determines which branch of the activity diagram is followed next because guitar and bass notes and drum notes are not created using the same methods. The software system will then remove unrecognized character notation for the given instrument, split the text file into measures and generate the notes of each measure. The notes in the measure are sorted in the order they appear. The measure is compiled into a list of measures. Once all notes have been generated and all measures have been compiled, the program runs a method that converts all of these objects into their equivalent MusicXML String representation and all of these strings are compiled to make the final MusicXML file. This demonstrates the connection between how the methods interact with each other to produce the output MusicXML file.

6.0 Front End Diagram UML Diagram

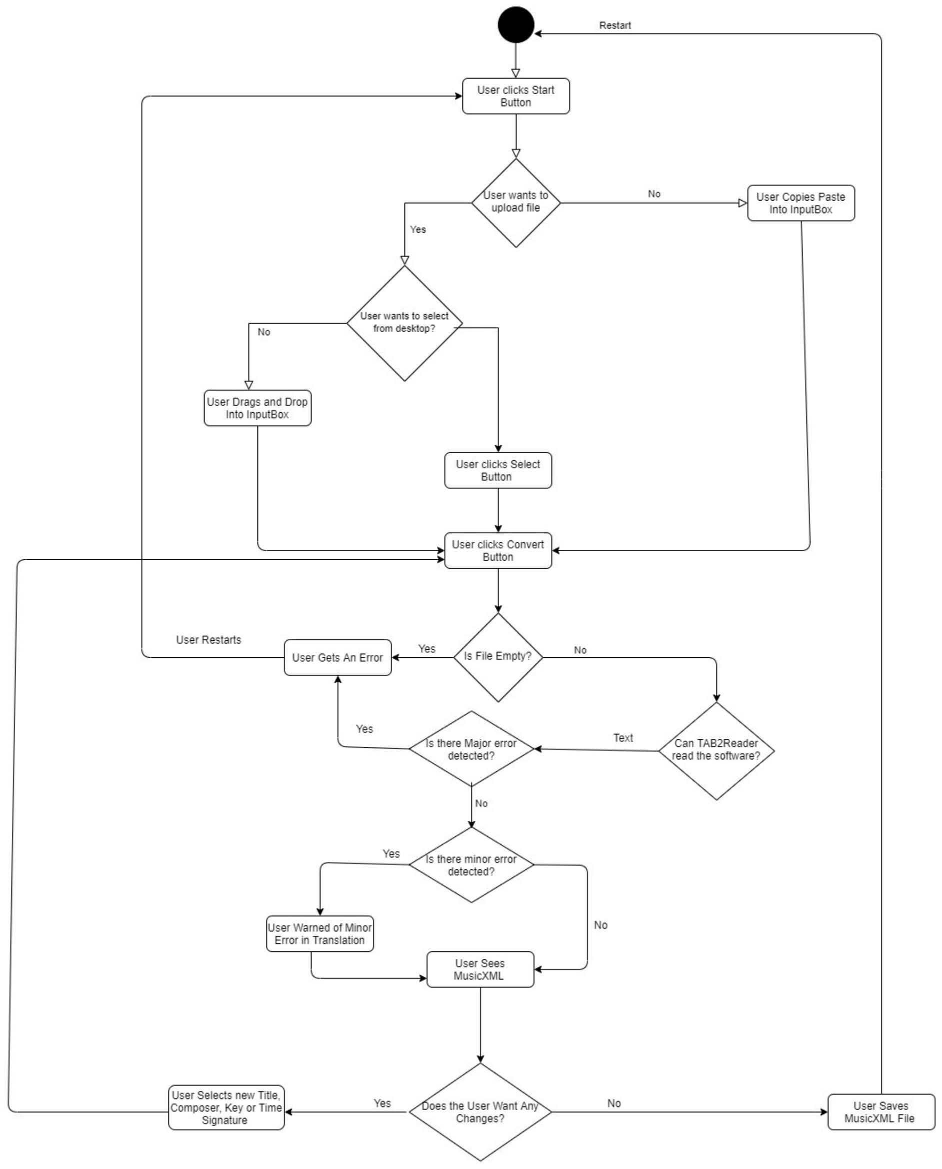
6.1 Front End Class Diagram


Abstract: Application
+void: start(final String[] args):
Extends
MainLaunch
+ void: main(String[] args)
Launch
Main
+ void: main(Start[] args)
+void: start(final String[] ar
Controller
+ TextArea: inputBox, outputBox, measureBox, measureView
+ Button: select, convert,save, featureButton, startButton, measureButton
+ Button: helpButton, timeSigButton, keyButton, titleButton, composerButton
+ TextField: measureText
+ TabReader: reader
+ void: viewMeasure()
+ void: saveInputFile()
+ void: convertClicked()
+ void: saveClicked()
+ void: dragFile()
+ void: select()
+ void: startClick()
+ void: helpClick()
+ void: selectComposer()
+ void: selectKey()
+ void: selectTimeSigniture()
+ void: selectTitle()
...

6.1.1 Interaction and Connection to Each Other

The Main class and MainLaunch methods interact when the application is initially launched. An instance of the Main class is used as an input parameter in the MainLaunch class. From there, the front end and GUI primarily operates by loading .fxml files located in the src/main/resources folder and by instantiating objects from the imported JavaFX packages, which were discussed in detail under the Front End Design section.

6.2 Front End Activity Diagram


Restart
User clicks Start Button
User wants to upload file
No
Yes
User Copies Paste Into InputBox
User wants to select from desktop?
No
User Drags and Drop Into InputBox
User clicks Select Button
User clicks Convert Button
User Restarts
User Gets An Error
Yes
Is File Empty?
No
Yes
Is there Major error detected?
Text
Can TAB2Reader read the software?
No
Yes
Is there minor error detected?
User Warned of Minor Error in Translation
No
User Sees MusicXML
User Selects new Title, Composer, Key or Time Signature
Yes
Does the User Want Any Changes?
No
User Saves MusicXML File

6.2.1 Interaction and Connection to Each Other

The activity diagram shows the connection of the front end and how the front end operates. It demonstrates the interaction between the user and the GUI. An example of this is if the user wants to upload a text file. The user can either have the option when selecting the file from their desktop or not. If the user decides not to upload the text file, they can copy and paste into the window. Furthermore, the activity diagram shows how the user will interact with the user interface and the different features the user interface displays.

7.0 Objects at Runtime

Clicking convert on the GUI sets in action a number of method calls in the back end as discussed in previous sections. This sequence of method calls instantiate the following objects:

- TabReader (1)
- TabError (1, which may be null if no error is detected)
- Measure (multiple)
- Notes (multiple)
- Pitch (multiple, or none if the instrument is drums)
- Unpitch (multiple, or none if the instrument is guitar or bass)

There are many other standard objects instantiated during the conversion procedure, such as HashMaps, ArrayLists and Strings. Ultimately, a String object is returned from the back end to the front end to be displayed in the GUI and saved with file extension .musicxml.

8.0 Glossary

Time Signature - Includes a numerator (element: beats) and denominator (element: beat-type)

Beats - Determines how many beats are in each measure

Beat-Type - Establishes the duration of one beat

Measure - Separates the music into segments, shown by a straight bar encasing notes

Note - a symbol that denotes musical sound, including it's pitch and duration

Note Duration - In a 4/4 time signature: whole (4 beats), half (2 beats), quarter (1 beat), eighth (1/2 beat), 16th (1/4 beat) and 32nd (1/8 beat)

Key - Numerical value is used to alter the pitch of an note up and down (numerical value is assigned based on the circle of fifths)

Division - Represents the number of division per quarter note and is used to indicate a note’s duration