Da Capo

Design Document

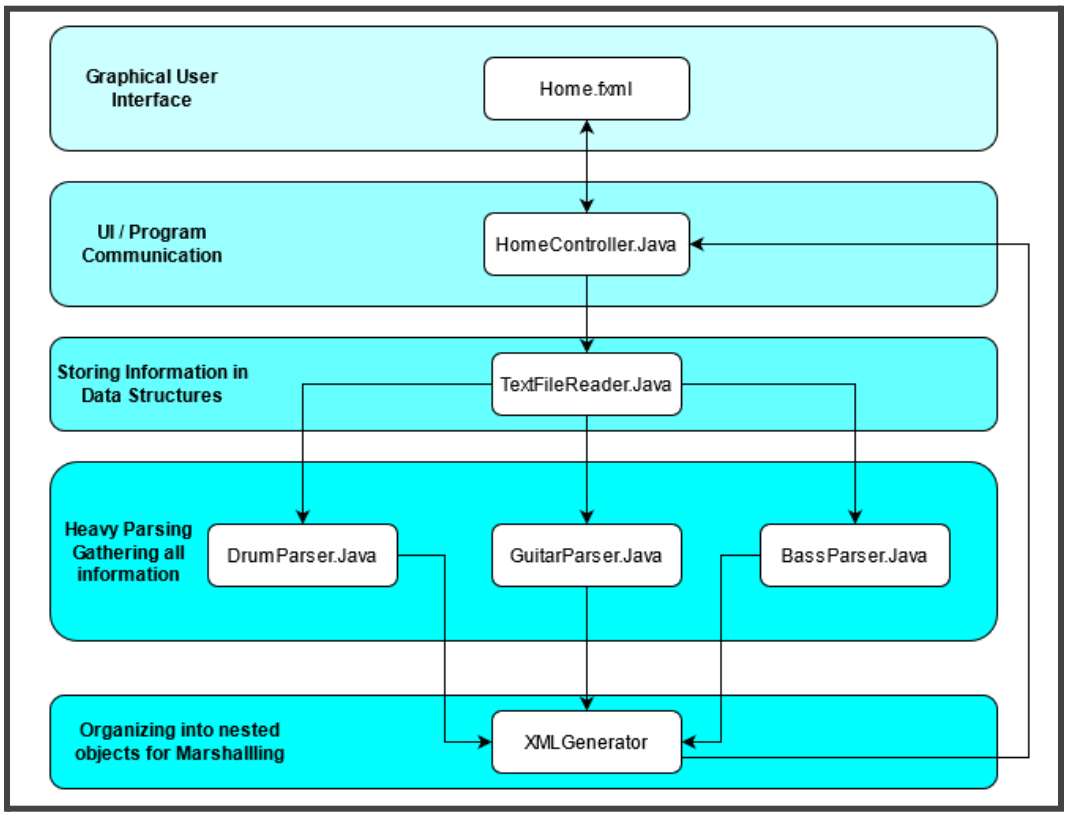
Version 1.1
Date: 04/13/2021

Gong-Fan (Billy) Bao - 213563150
Syed Quadri - 216204554
Elijah Nnorom - 214502504
Miguel Mesa Gonzalez - 216138323
Ayub Osman Ali - 217612847

Table of Contents

1. System Overview

1.1 System Overview



The System that Da Capo uses to operate is composed of five major Sections:

1. **Graphical User Interface**
   The Graphical User Interface involves all the Components the User is able to communicate with in order to use the System. Any processes must be initiated from this section. The Home.fxml Class is solely responsible for the visual appearance of the System and works in close conjunction with its Controller.
2. **UI/ Program Communication**
   The Communication between the User Interface and Program is crucial in linking what the User Initiates and what processes the program executes. The HomeController Class is responsible for this communication and serves as the brain of the System's operations.

3. **Error Detection and Information Storage**

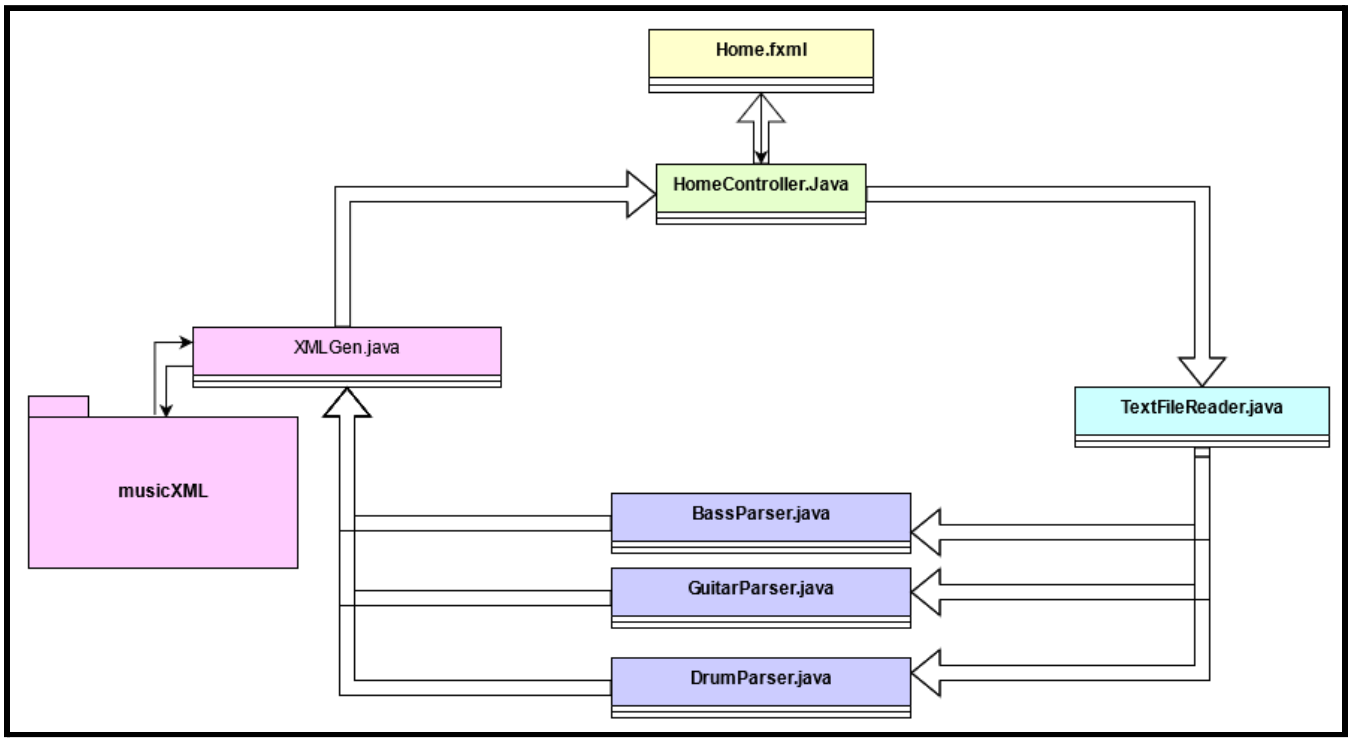
   The Information received from the user is stored in this area and cleaned for further processing down the line. This section of the System is the last line of defense to prevent invalid information from reaching the more vulnerable sections of the code after where it will be prone to crashing with slightly unorganized input. The TextFileReader Class is responsible for digesting the information and for scrutinizing the passed information to throw errors that the User Interface may display.

4. **Heavy Parsers**

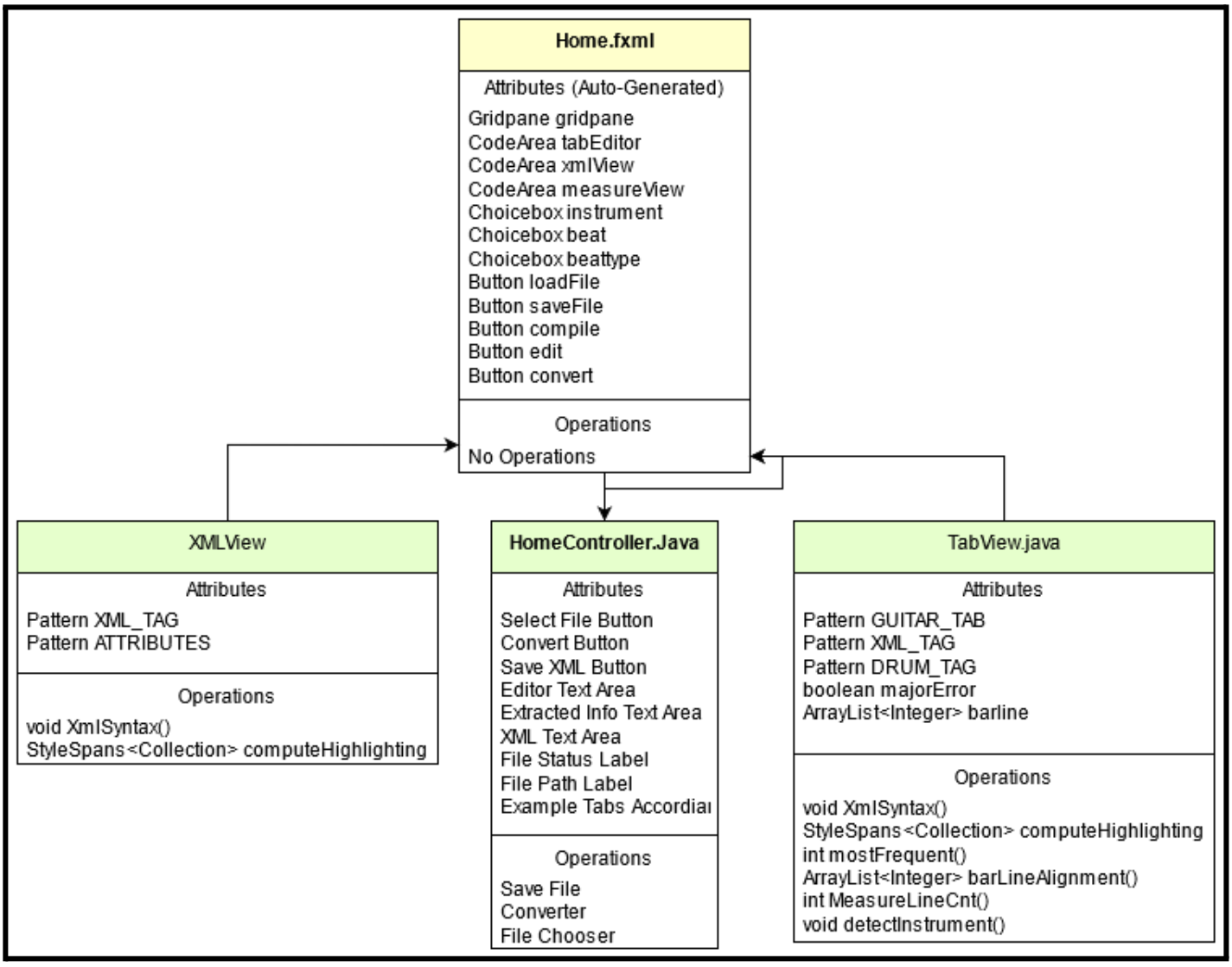
   This is where the core of the System lies. Any information that is passed on to this section will be processed in detail and organized into the desired output for marshalling in the next section. The System no longer communicates with the user after this step. The classes responsible for this section are the GuitarParser, BassParser and DrumParser.

5. **Marshaller**

   The Marshalling is the final step of the System, where the information gained from the heavy parsers is further organized into the exact sections required for the appropriate XML output. The XML Generator Class works with a plethora of Schema classes to create a large group of nested objects which is then marshalled and sent to the User Interface Controller as the end product for the user.



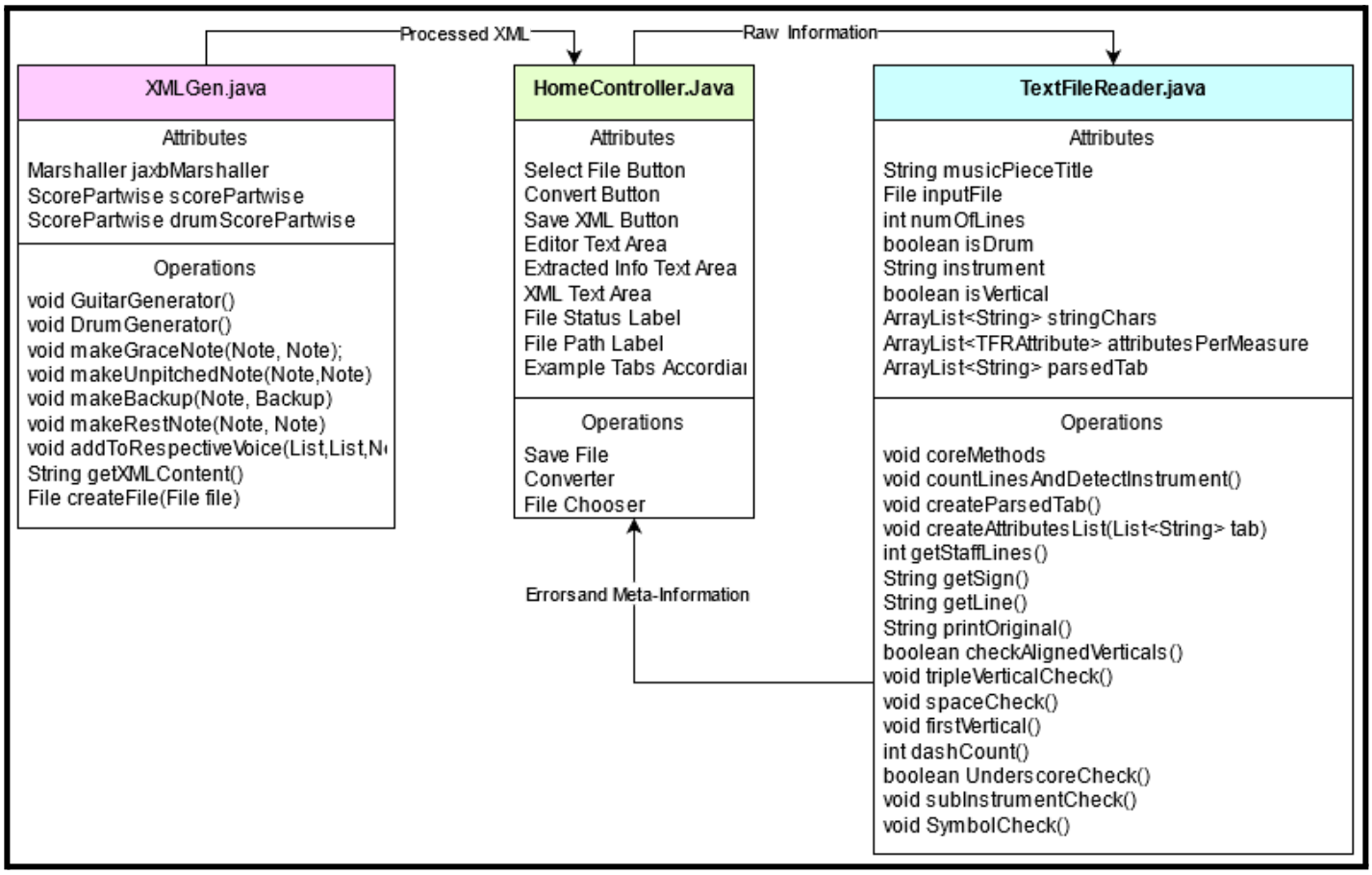
1.1 Graphical User Interface Class Diagrams



The GUI Classes

There are three classes that function alongside the HomeController Class which maintains lines to the back end of the software. These three classes split up the overall task of the graphical user interface. While home.fxml contains all the information about the appearance of the Interface, the TabView and XMLView support the use of library of richtextfx that allows the format of the input to be styled using Regex to display errors to the user and format the tablature.

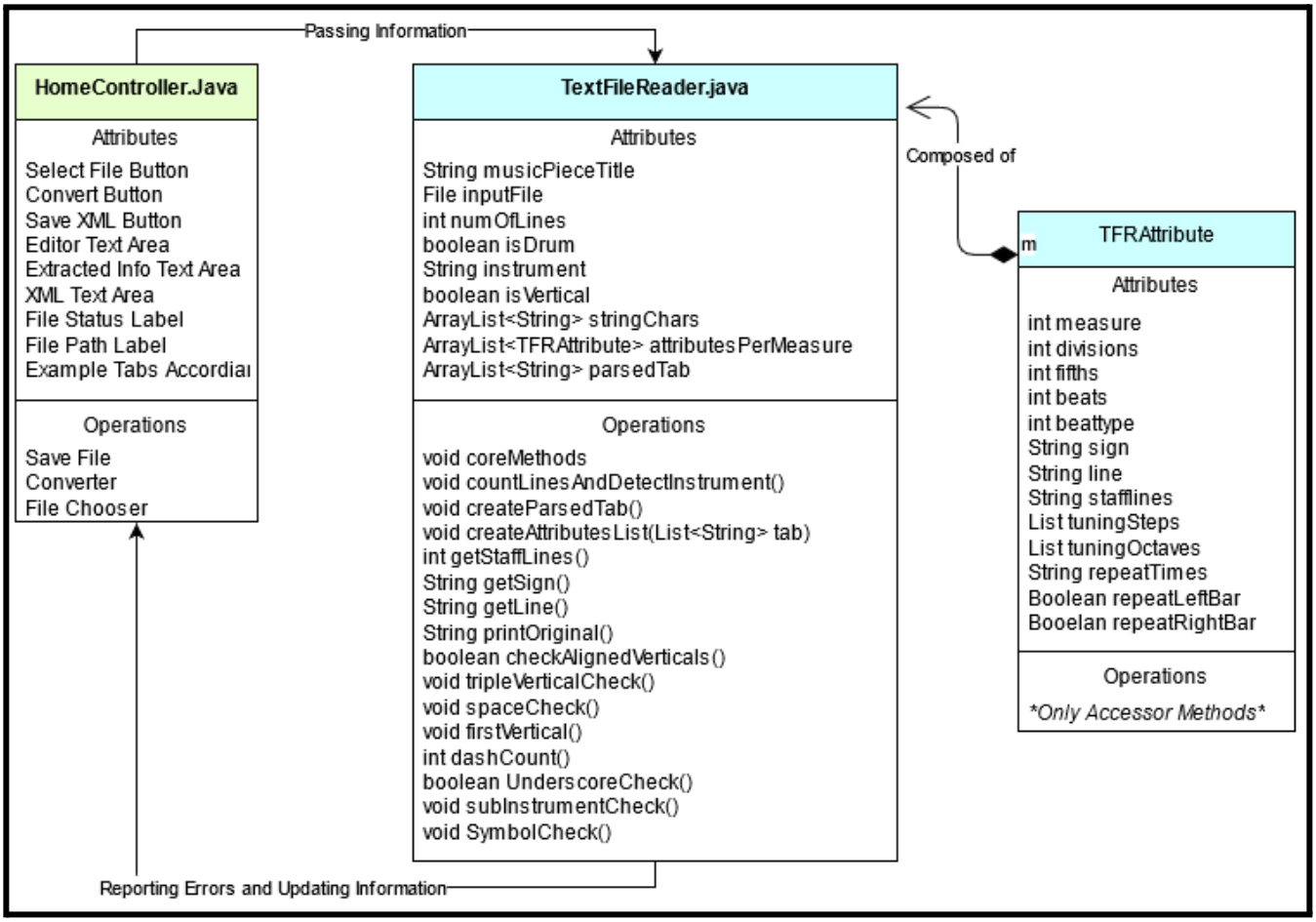
1.2 HomeController Class Diagrams



The HomeController Class

The HomeController Class is the hub of all the activity between the User Interface and the Bank End executions. It contains several global variables responsible for saving information given by the user on the GUI as well as displaying it if necessary. The Class contains a plethora of methods that are used to allow action events such as the click of a button or the drag and drop of a file. It contains all the necessary object calls to be able to communicate errors to the user as well as pull back information from the TextFile reader classes to allow the user to view. The HomeController also functions as the gateway for the completed tablature to come back from. In essence, any functionality that needs to serve as the brain of the software must be conducted here as this is the main file that coordinates the flow of all the other files as soon as the main class begins the instantiation of the software.

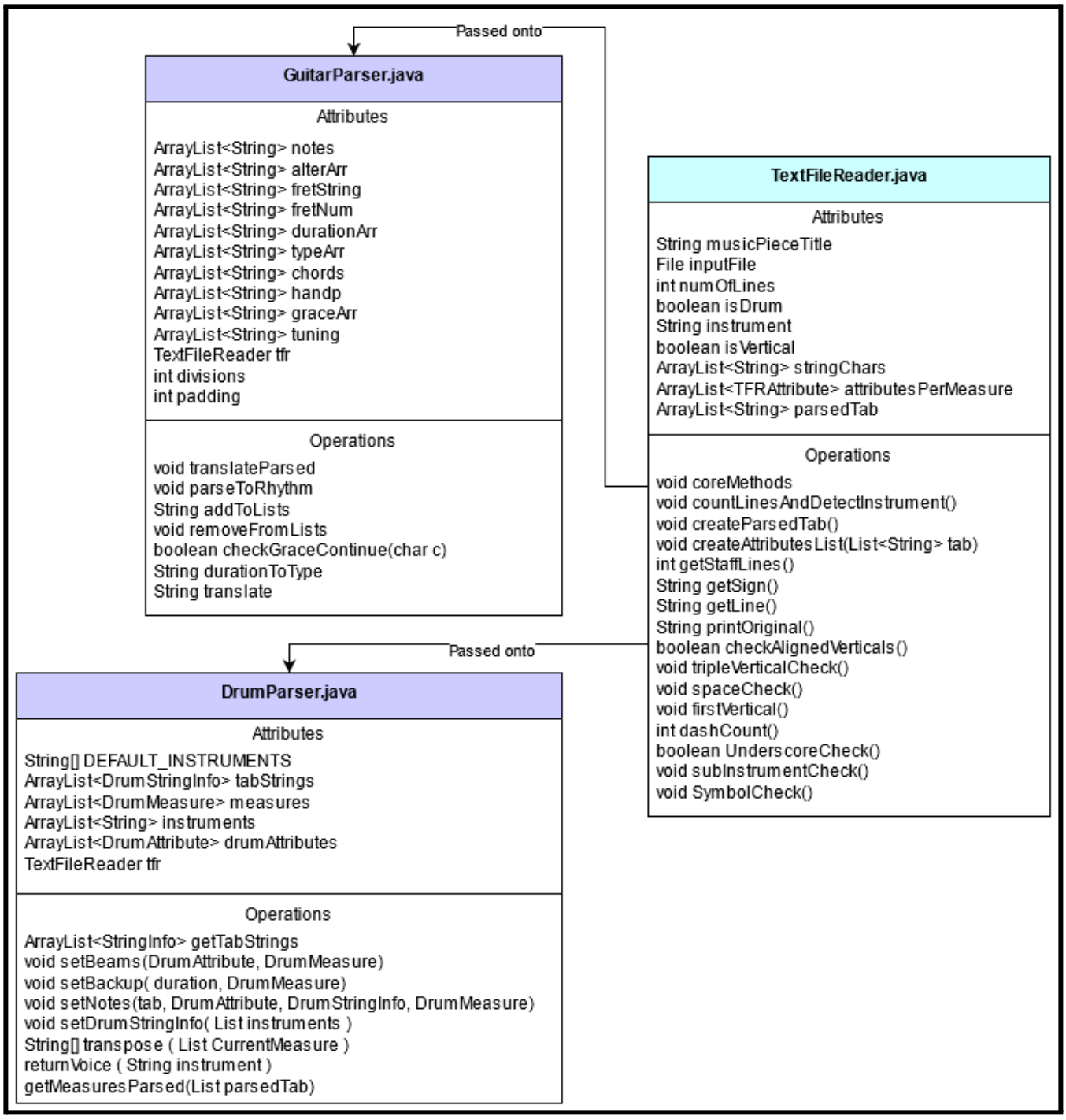
1.3 TextFileReader Class Diagrams



The TextFileReader Class

The TextFileReader Class is the Filter and Information Evaluation Section of the System. Above it, is the HomeController which is responsible for gathering information from the user and passing it along. Once this raw information is received, the TextFileReader immediately parses it, gathering basic information that will allow the rest of the System to function along the right path. Some examples include the number of lines the tablature as well as what instrument the tablature is for. This information is necessary in order for the parsers to function correctly. The decision of which Class to utilise next for the heavy parsing is decided here. If a guitar tablature is discovered, the GuitarParser class is called and likewise for Drums and Bass, where the DrumParser and BassParser are called respectively. On top of gathering this information, the TextFileReader Class also has a dual function of error handling. While gathering and processing the meta-information, it simultaneously runs error checking processes which allows the System to communicate cases of invalid input back to the User Interface Controller, the HomeController in the diagram above.

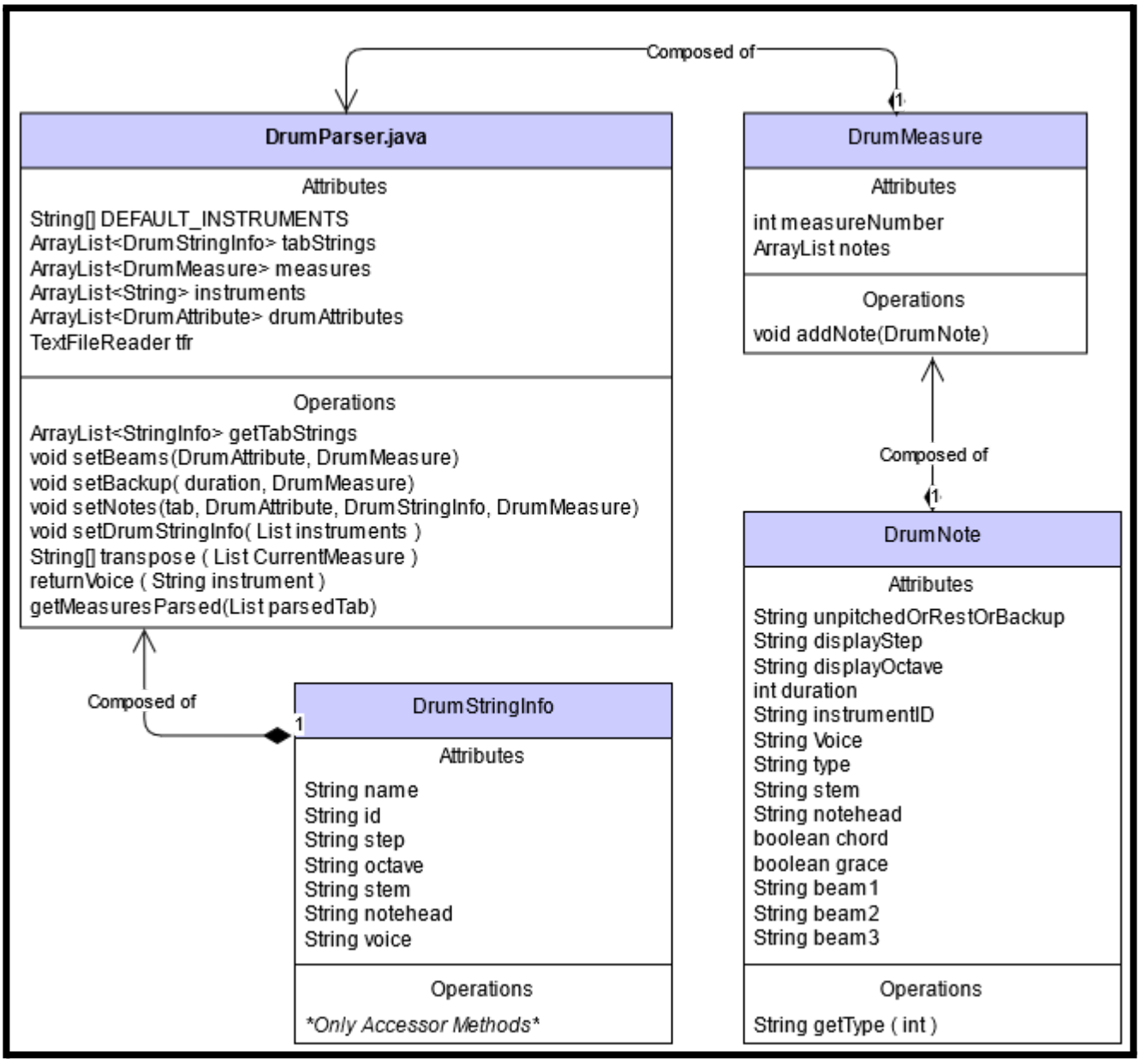
1.4 Parser Class Diagram



Parser Classes

The two parser classes for drums and guitar are responsible for the heavy parsing and extracting of information that has been formatted and passed down from TextFileReader. Currently the GuitarParser is able to process Bass Guitar, due to the similarity in the two instruments.

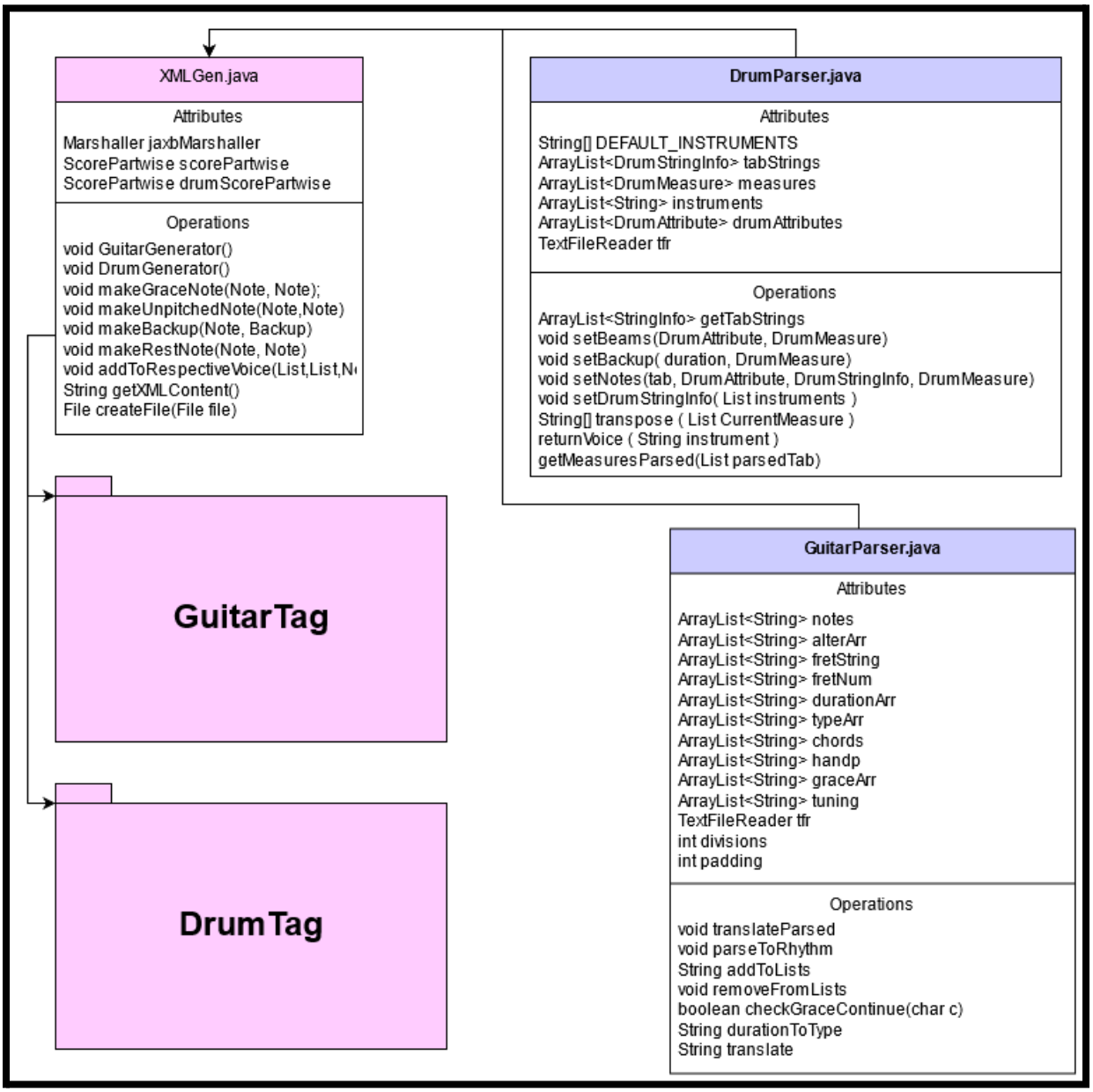
1.5 Drum Parser Class Diagram



The DrumParser Class

The DrumParser Class contains 3 Subclasses that it uses to store parsed information. While the Guitar parser and Bass parser utilize arrays to store gathered information, the Drum parser stores the information into object attributes and utilizes aggregation to store multiple Note objects into Measure objects. Information regarding instruments and lines is stored in another set of objects of the StringInfo class.

1.6 XML Generator Class and Tag Packages

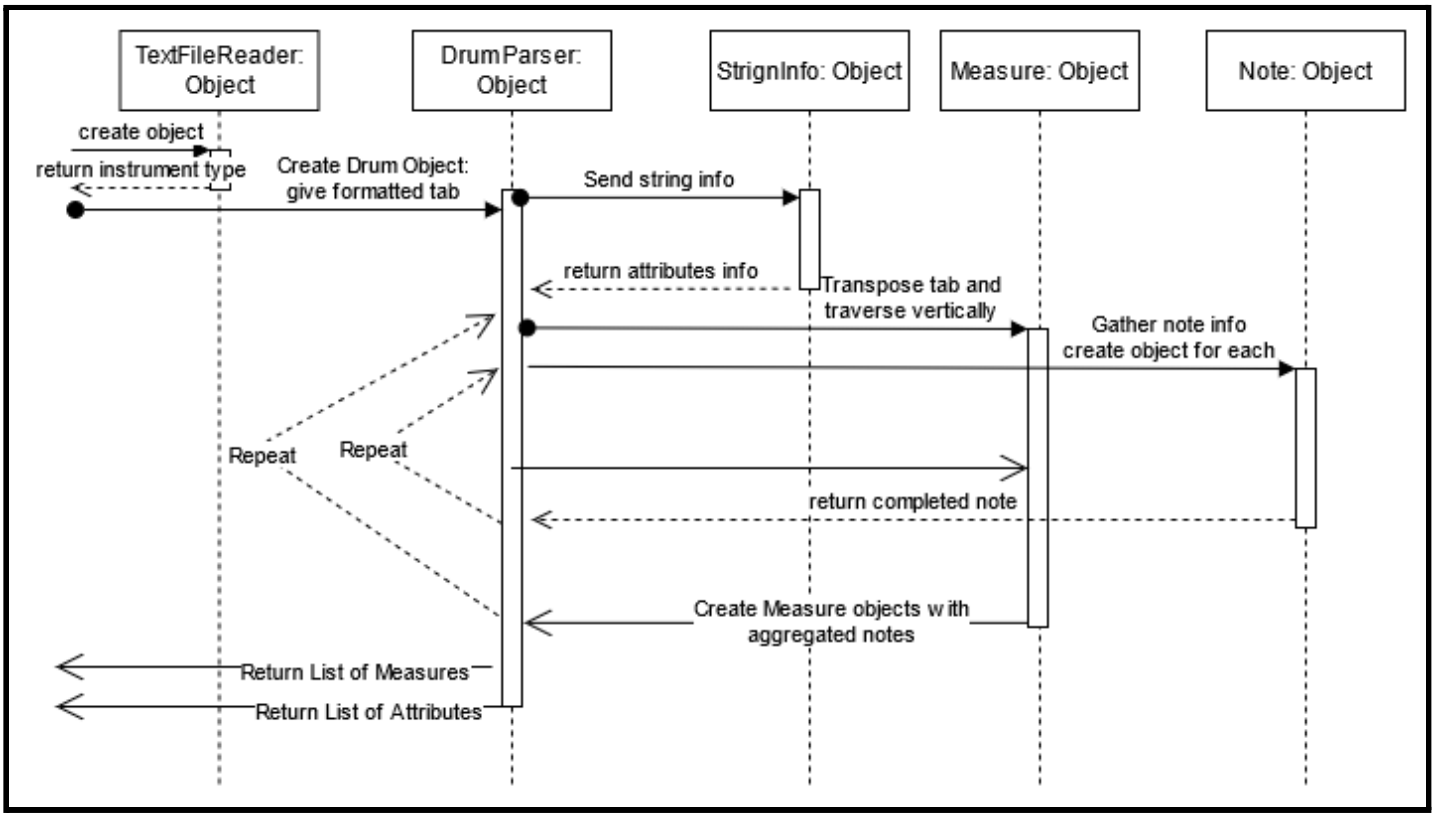


The XMLGenerator Class

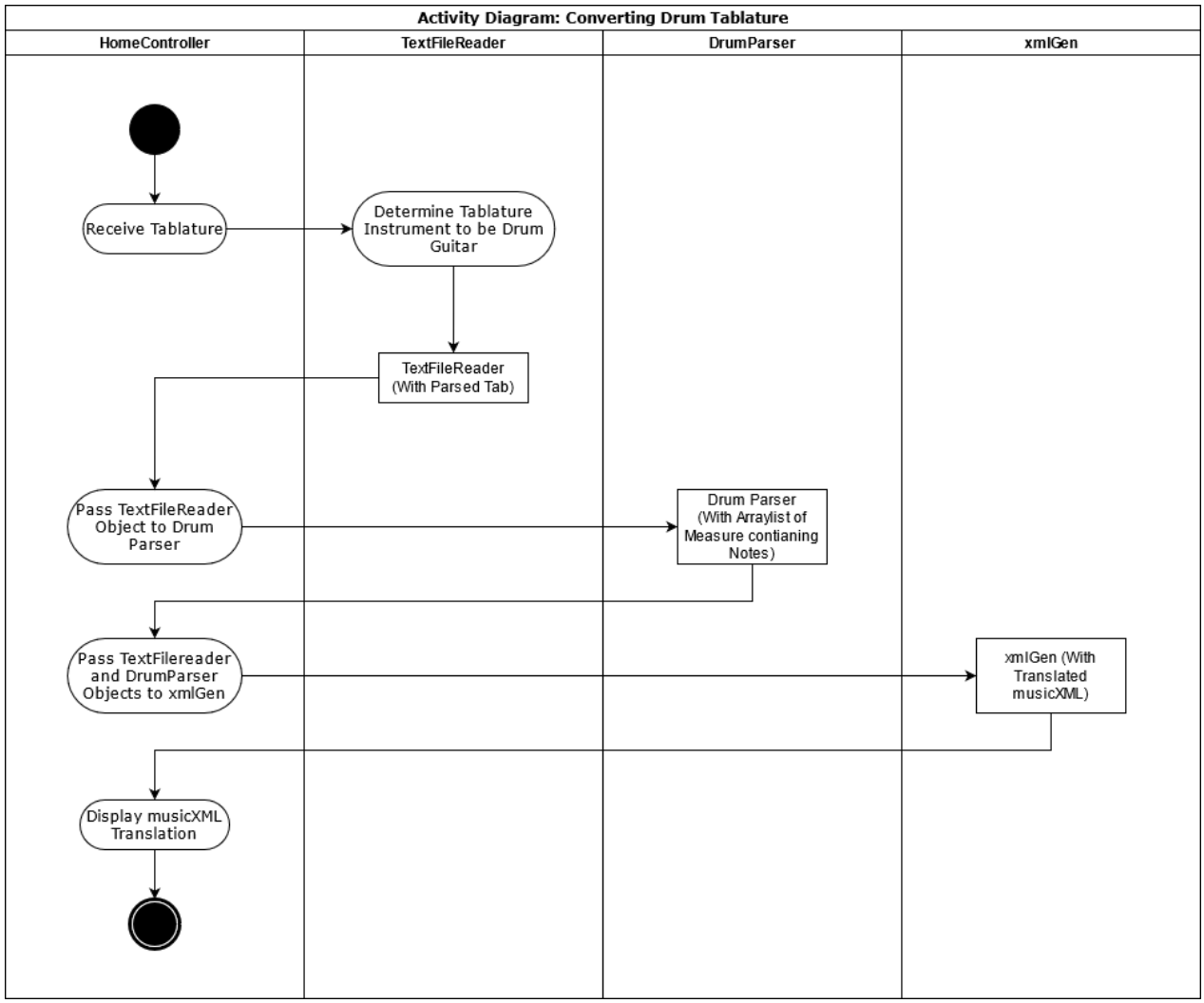
The XMLGenerator Class is the final processing step of the System. It Works alongside a package of musicXML Tag Classes that are used to create a large set of nested objects based on information that is passed from the Parser classes above it. Once it has produced the Marshalled output, the XML Generator will pass it along to HomeController that will perform basic operations such as printing it out for the user to see or for saving it into a file.

2. Parser Sequence Overviews

2.1 Parsing Drum Tablature

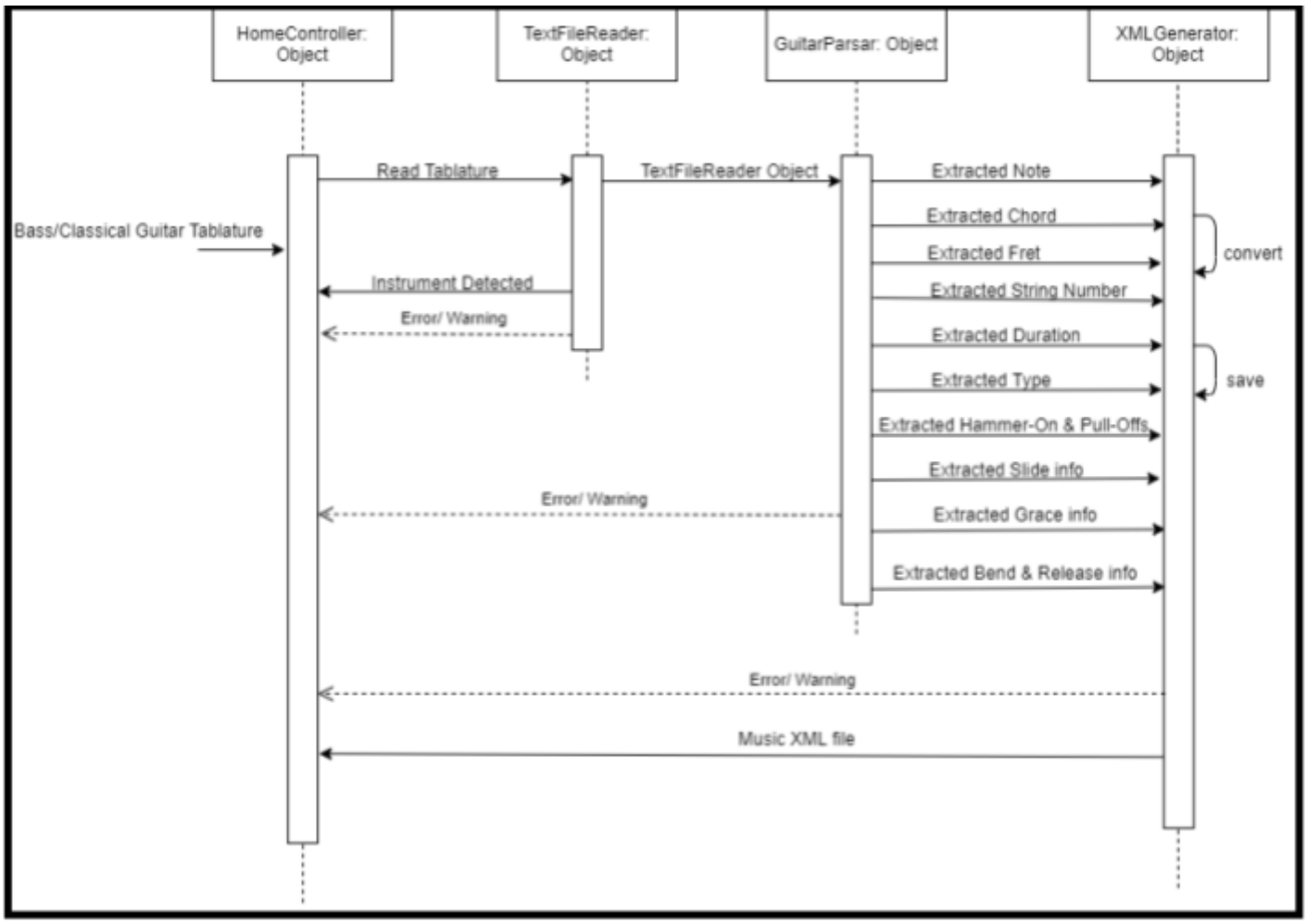


The Sequence Diagram above describes the process by which Da Capo extracts information from a given Drum tablature. An object of the TextFileReader Class is first inserted and the internal processes are executed to clean the information and determine the instrument. If the instrument detected is a Drum, we continue along the pathway, passing the information to a DrumParser object. The Drum parser then iterates over the information creating StringOnfo objects that are later called in the iterative process of several Measure objects with several nested Note objects . By the time the iterative process is complete , all the information has been extracted from the passed input into a list of attributes and Measures that is passed on to the XMl Generator.

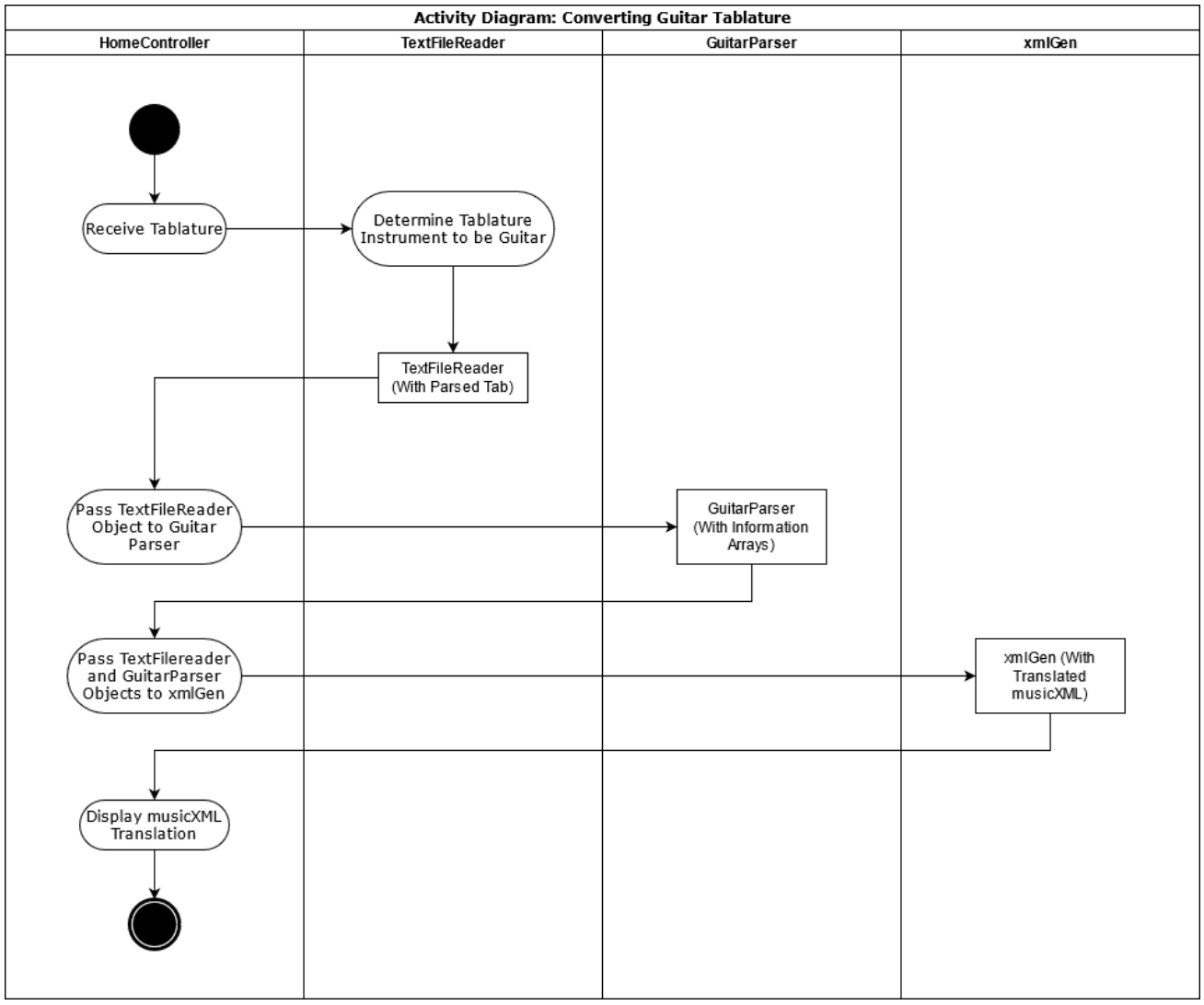


The Activity Diagram above describes the activities that occur within the System, when it is converting Drum tablature. First, the HomeController class receives the tablature that will be converted, which is then passed onto TextFileReader. TexteFileReader generates a TextFilereader object with a Parsed Tab, which is then passed onto the DrumParser, to generate a DrumParser object with an ArrayList with Measures containing Notes. Next, both the TextFileReader and DrumParser objects are passed onto the xmlGen class, which generates the translated musicXML, which is then passed back to HomeController. Finally, HomeController displays the translated musicXML to the User.

2.2 Parsing Guitar Tablature

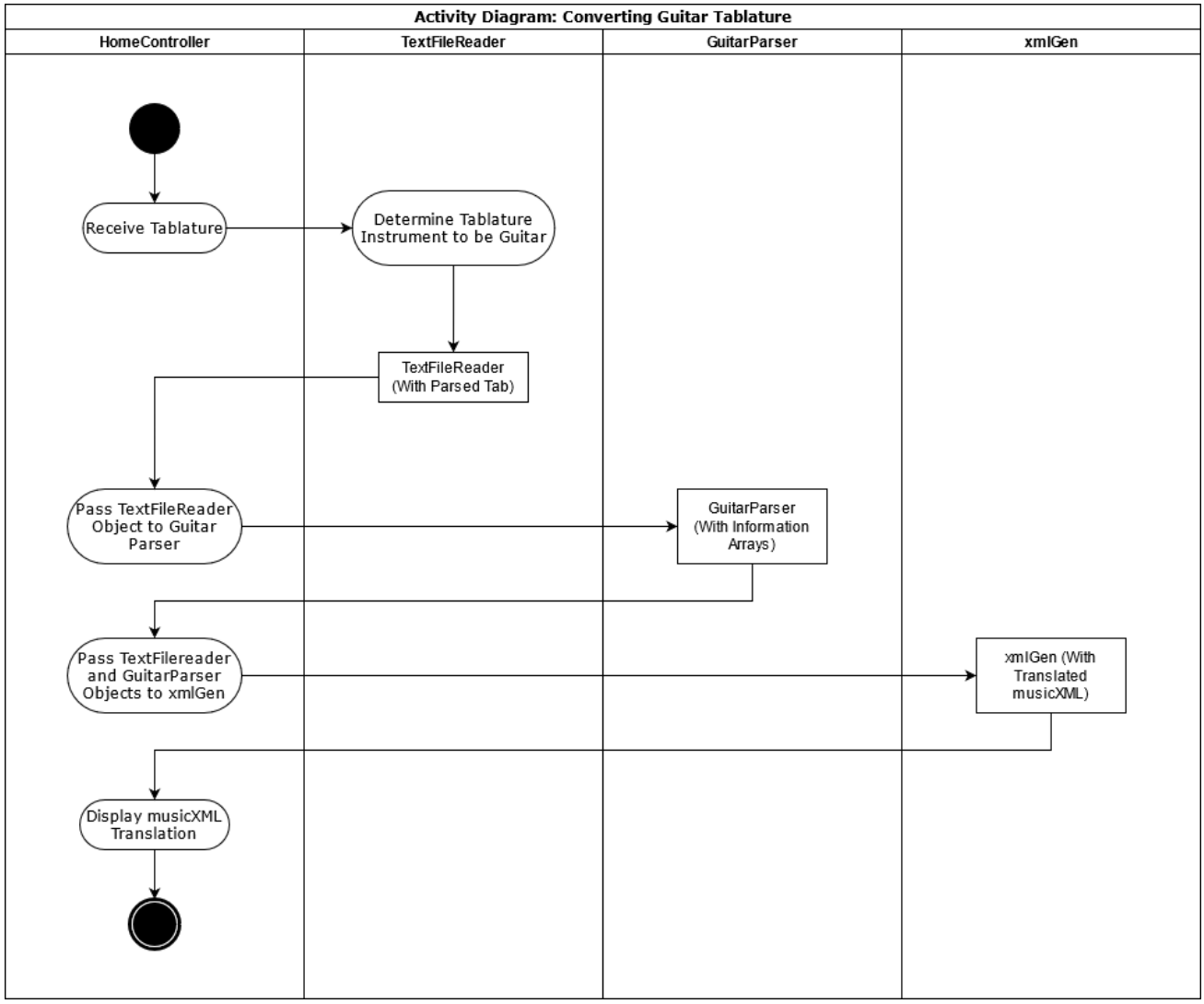


The sequence diagram above illustrates the end-to-end interaction between objects that perform the translation of Classical/Bass Guitar to musicXML. In our System, the Bass and Classical Guitar are identified as two different instruments but the process of generating a musicXML file is similar for both instruments. Input is provided to the HomeController through the GUI and after it is verified to be a Tablature of a valid instrument, the TextFileReader converts it to a format that allows the GuitarParser to extract information. The GuitarParser then extracts all the information needed to generate the appropriate musicXML file which is sent to the XMLGenerator who generates the musicXML. The musicXML is then sent to the HomeController and subsequently displayed on the GUI, if every step in this process is successful. If something went wrong along the line in any stage of the sequence, depending on what the issue is, the appropriate error message or warning message is sent back to the HomeController and this message is displayed on the GUI.



The Activity Diagram above describes the activities that occur within the System, when it is converting Guitar tablature. First, the HomeController class receives the tablature that will be converted, which is then passed onto TextFileReader. TexteFileReader generates a TextFilereader object with a Parsed Tab and Measure attributes, which is then passed onto the GuitarParser, to generate a GuitarParser object with arrays containing Note information. Next, both the TextFileReader and GuitarParser objects are passed onto the xmlGen class, which generates the translated musicXML, which is then passed back to HomeController. Finally, HomeController displays the translated musicXML to the User.

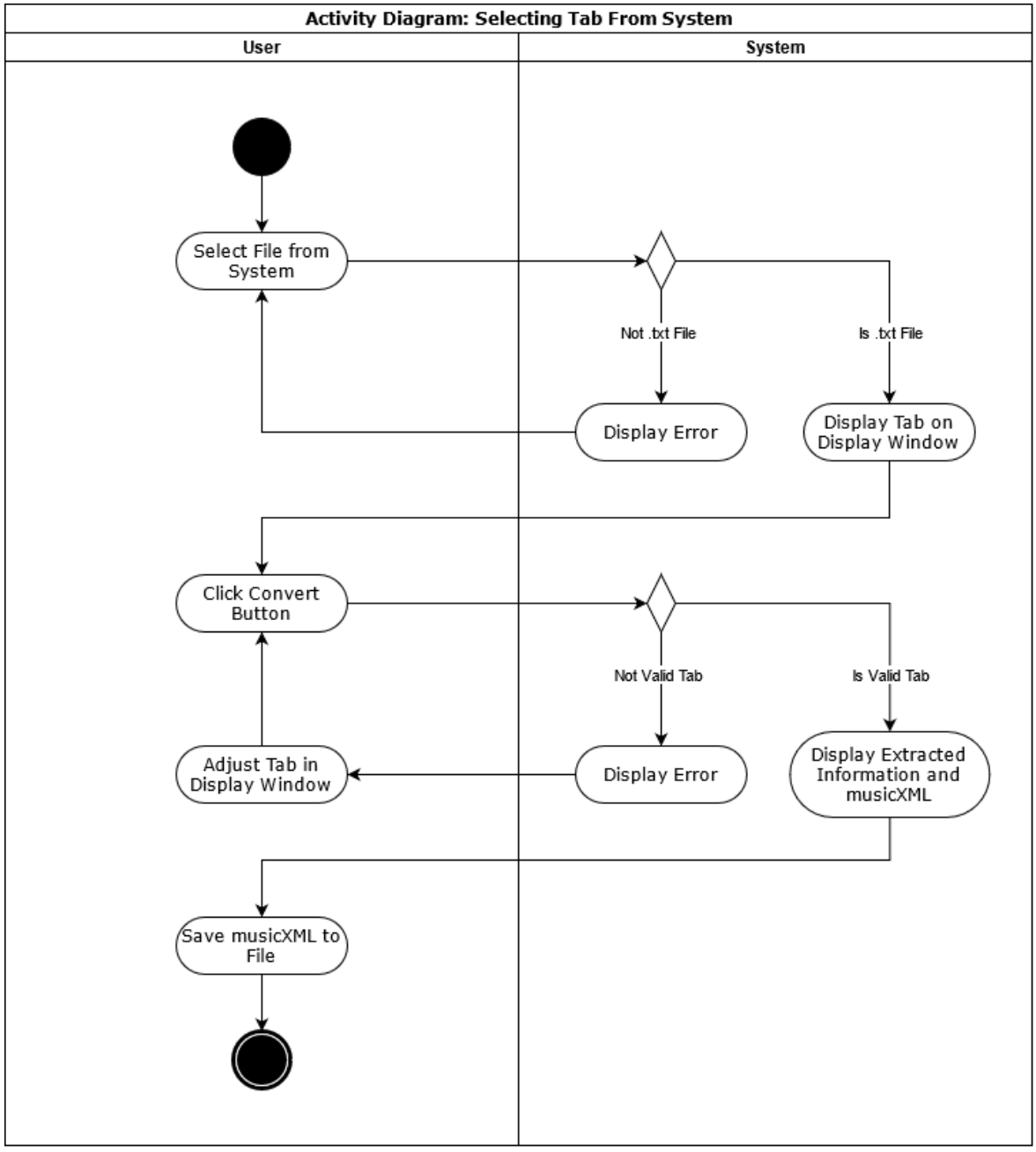
2.3 Parsing Bass Tablature



The Activity Diagram above describes the activities that occur within the System, when it is converting Bass tablature. First, the HomeController class receives the tablature that will be converted, which is then passed onto TextFileReader. TexteFileReader generates a TextFilereader object with a Parsed Tab and Measure attributes, which is then passed onto the GuitarParser, to generate a GuitarParser object with arrays containing Note information. Next, both the TextFileReader and GuitarParser objects are passed onto the xmlGen class, which generates the translated musicXML, which is then passed back to HomeController. Finally, HomeController displays the translated musicXML to the User.

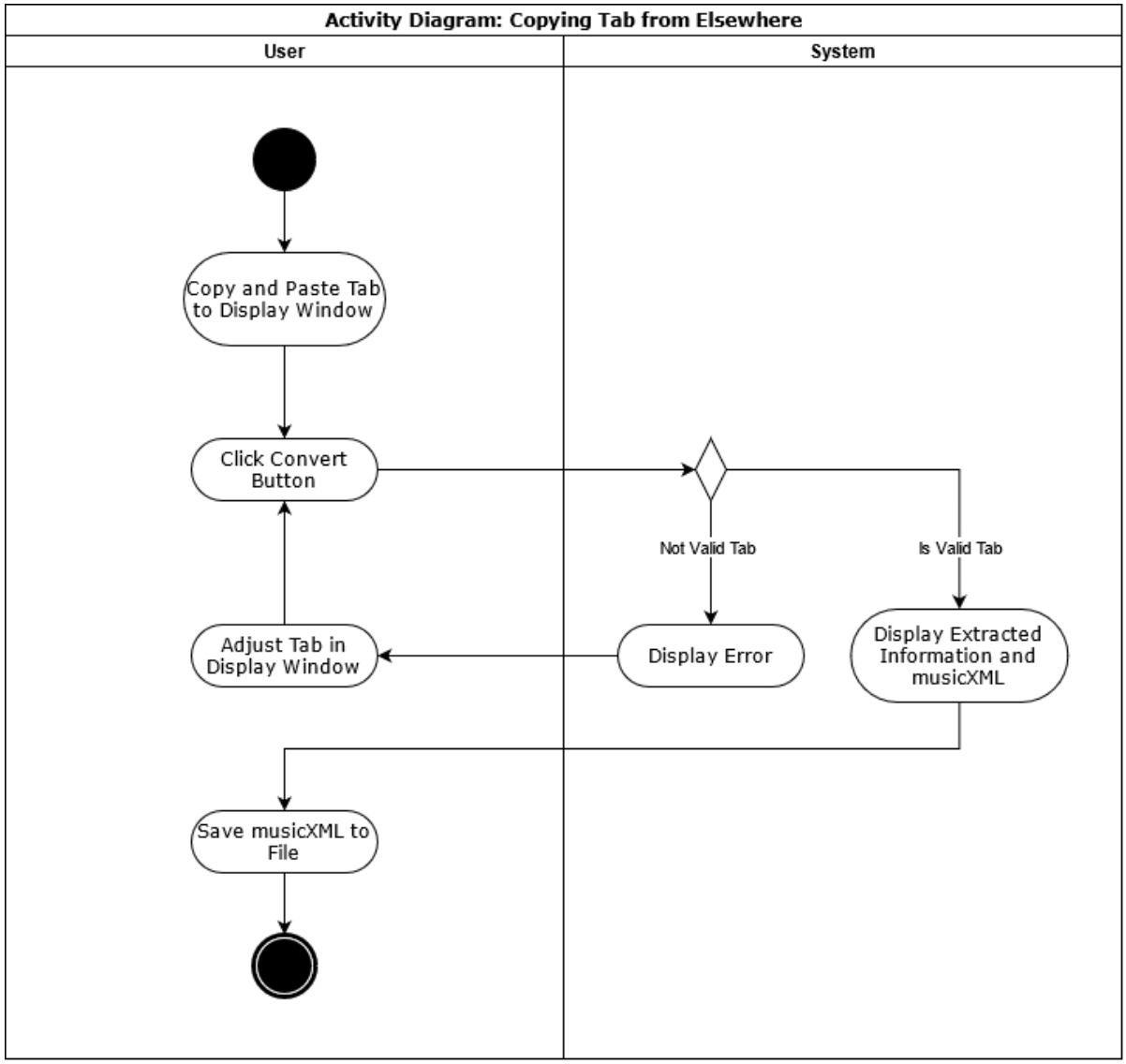
3. Use Cases

3.1 Selecting Tab from System



The Activity Diagram above describes the activities that occur between the User and the System, when the User wants to translate a tab from a .txt file on their computer.

3.2 Copying and Pasting Tab from Elsewhere



The Activity Diagram above describes the activities that occur between the User and the System, when the User wants to translate a tab that has been copy and pasted from somewhere else, such as a website.

4. Maintenance Scenarios

4.1 Supporting a New Instrument

4.1.1 Class/File to Modify

- New instrument parser class must be created
- TextFileReader.java
- xmlGen.java

4.1.2 Steps to Modify:

- New instrument parser class:
    - This class will have to be created
- TextFileReader.java
    - Will need a new section in detectInstrument
    - Will need a new case for the staffLines, sign and line methods
    - May need modifications in the countLines scanner similar in order to differentiate it from other instruments
- xmlGen.java
    - Will need its own set of attributes to be imported and set

4.2 Supporting Bass with More Strings / Guitar with Less Strings

4.2.1 Class/File to Modify:

- TextFileReader.java
- GuitarParser.java

4.2.2 Steps to Modify:

- TextFileReader.java
    - Will modifications to detectInstrument so that it can differentiate between a bass with the same amount of lines as a guitar or drum.
- GuitarParser.java
    - May need some modifications in order to make it more robust and ensure the tuning isn't hard coded

4.3 Supporting Graphical User Interface Changes

4.3.1 Class/File to Modify:

- homeController.java
- home.fxml

4.3.2 Steps to Modify:

- Home.fxml
    - Any new GUI element is created and added in this file.
- HomeController.java
    - An event handler should be made.

4.4 Supporting new Tablature Symbols

4.4.1 Class/File to Modify:

- TextFileReader.java

4.4.2 Steps to Modify:

- TextFileReader.java
    - In order to support new symbols such as “@” or “v” the class will need a method that will catch these symbols and determine how they will impact the parsers, the TFRattribute classes and the other methods in TextFileReader such as dashCount.

4.5 Supporting additional Meta-Information

4.5.1 Class/File to Modify:

- Home.fxml
- Homecontroller.java
- TextFileReader.java
- xmlGen.java

4.5.2 Steps to Modify:

- Home.fxml

    - A new GUI element should be made so the user can input artist and date or any other meta-information
- Homecontroller.java
    - An event handler should be made to take the information from the interface and give it to the TextFileReader.java to set as an attribute and pass it to xmlGen.java
- TextFileReader.java
    - A new attribute should be made to store the meta-information and pass it to xmlGen.java
- xmlGen.java
    - A new method can be made to receive and output this meta-information

4.6 Supporting additional MusicXML Tags

4.6.1 Class/File to Modify:

- xmlGen.java
- TextFileReader.java
- GuitarParser.java OR DrumParser.java

4.6.2 Steps to Modify:

- xmlGen.java
    - Will need to call the appropriate classes and create the additional tags.
- TextFilereader.java
    - May need to have an array of new attributes, related to the additional tags, which will be used by GuitarParser.java, DrumParser.java or xmlGen.java
- GuitarParser.java OR DrumParser.java
    - May need to have an array of new attributes, related to the additional tags, which will be used by xmlGen.java

4.7 Supporting Modifiable Time Signature

4.7.1 Class/File to be modified:

- Home.fxml
- HomeController.java
- TextFileReader.java
- GuitarParser.java
- DrumParser.java

4.7.2 How will each Class/File need to be modified:

- Home.fxml
    - Any new UI element to handle the additional input is created and added in this file.
- HomeController.java
    - An event handler that will modify the value of the time signature of the given measure. This modification needs to happen in the TextFileReader.
- TextFileReader.java
    - This class must now be responsible for detecting and passing the time signature to the parsers.
- GuitarParser.java, DrumParser.java, BassParser.java
    - The following classes must now be able to receive time signatures passed by TextFileReader.

4.8 Supporting In-Program Music Player

4.8.1 Class/File to Modify:

- Home.fxml
- HomeController.java
- New class will have to be created
- xmlGen.java

4.8.2 Steps to Modify:

- New classes
    - A new class to play mp3 music files will need to be constructed

    - A new class to convert the xml file into an mp3 file will need to be constructed
- Home.fxml
    - A new GUI element must be made to activate the player
- HomeController.java
    - This will handle the input from the GUI element and tell the new class to play the file
- xmlGen.java
    - The xmlGen will need to pass the xml file through these new classes to play the tab