Design Document

V2 - April 13

By EECS2311 Team 4

- Patchanon Suepai
- Suha Siddiqui
- Nobaiha Zaman Rayta
- Zhilong Lin
- Martin Brejniak

Table Of Contents

1. Introduction

This document will go over all of the classes/methods that make up our program, along with how they work together. Maintenance scenarios will also be provided. Detailed class and sequence diagrams can be found throughout. The visualizations provided should help gain an architectural perspective of the software program developed. Should you have any further questions or concerns, feel free to reach us through the email provided in this document. (EECS2311Team4@gmail.com).

2. Overview of System

The system consists of classes that split the system up into components that deal with drums and components that deal with guitar/bass. All the control flow happens within the main class.

In the main class, there are a few main functions that control the flow of the program. Once everything in the GUI has been resolved, it will call upon a function named start, which starts the flow of the conversion. The start function will then either call upon a drum or guitar parsing function which creates internal representations of guitar/drum notes. These notes are then passed to their respective XML parsing function which takes the notes and reads the values and returns an XML string. This XML string is then passed back to the GUI for the user to read/save.

3. Classes

3.1. Main

3.1.1. Description

The main class of the programs. It allows for all of the other classes to work together. The whole program can be started up by just running this class.

3.1.2. Methods

public static boolean fileChecker(String file)
Function is called to check if the input file is a text file.

public static Object[] fileParser(String file) throws FileNotFoundException

Once checks pass, this function is called to parse the input, returning an array of the important extracted information.

public static ArrayList<GuitarNote> guitarNoteParser(ArrayList<String> noteArray)
Function takes an arraylist of notes given from fileParser and converts the individual raw text notes into GuitarNote objects which represent guitar notes and returns an arraylist of that.

public static ArrayList<DrumNote> drumNoteParser(ArrayList<String> noteArray)
Function takes an arraylist of notes given from fileParser and converts the individual raw text notes into DrumNote objects which represent guitar notes and returns an arraylist of that.

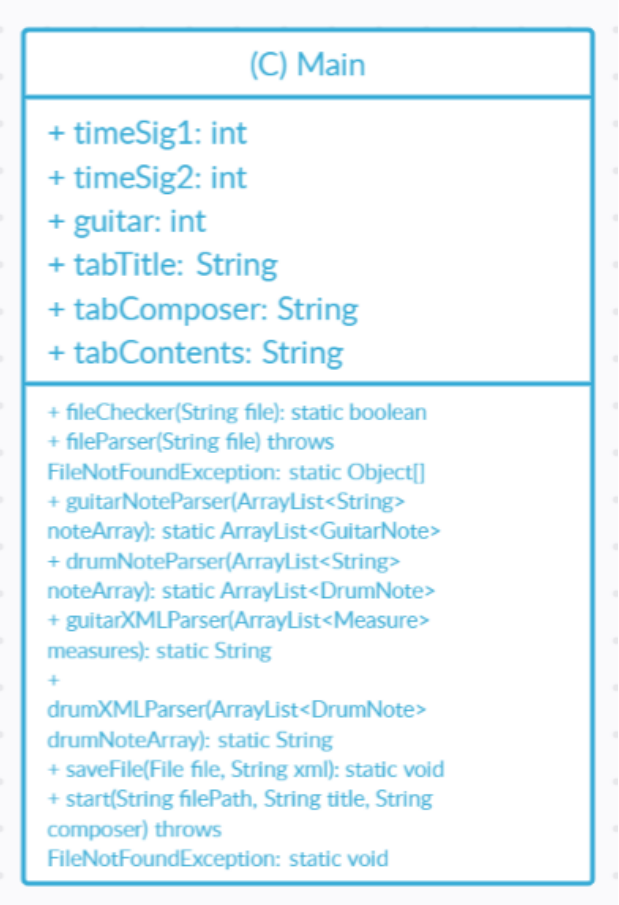
public static String guitarXMLParser(ArrayList<Measure> measures)
Function takes an arraylist of measures and converts it into an XML. Returns a string which is the XML

public static String drumXMLParser(ArrayList<DrumNote> drumNoteArray)
Takes all the information gathered from the drum tablature and makes a MusicXML file with it.

public static void saveFile(File file, String xml)
Function saves the file onto the user’s computer. Takes the XML string and a file name.

public static void start(String filePath, String title, String composer) throws FileNotFoundException
Main function, called from the GUI to start the conversion process

3.1.3. Class diagram



3.2. GuitarNote

3.2.1. Description

The internal representation of a guitar note. Each note inside the text tablature gets converted into this object.

3.2.2. Methods

public GuitarNote(int measure, int noteNumber, int stringValue, String noteValue)
The default constructor, it requires the measure number, note number, string number, and the note value.

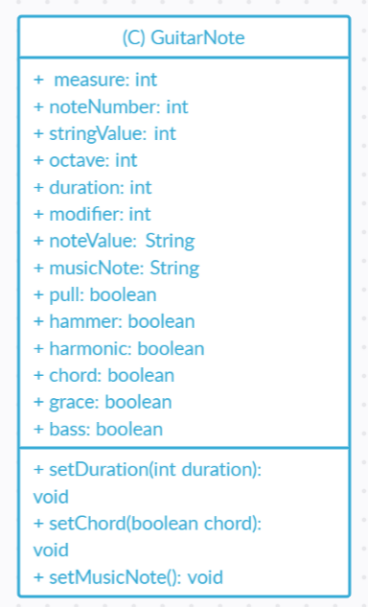
public void setDuration(int duration)

Sets the duration of the note.

public void setChord(boolean chord)
Sets if the note is in a chord or not

public void setMusicNote()
Converts from the internal representation of note into the musical representation

3.2.3. Class diagram



3.3. Measure

3.3.1. Description

The internal representation of a measure. Each measure inside the text tablature will have a corresponding measure object associated with it. Each measure contains information about the measure as well as an array of either guitar or drum notes.

3.3.2. Methods

public Measure(int elementMax, int measureNum)

Default constructor, requires the maximum amount of elements in the measure, and the measure number.

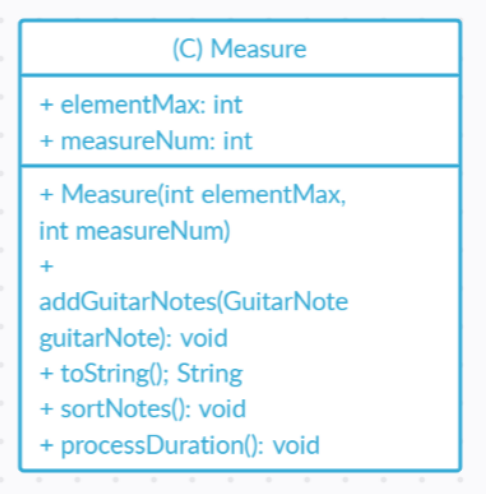
public void addGuitarNotes(GuitarNote guitarNote)
Adds a guitar note into this measure

sortNotes()
Sorts the guitar notes inside this measure in ascending order.

processDuration()
Finds the duration of each guitar note inside the measure.

public void setNoteType()
Converts each note in the array from the numerical duration to a string representation to be used in XML parsing.

3.3.3. Class diagram



3.4. DrumNote

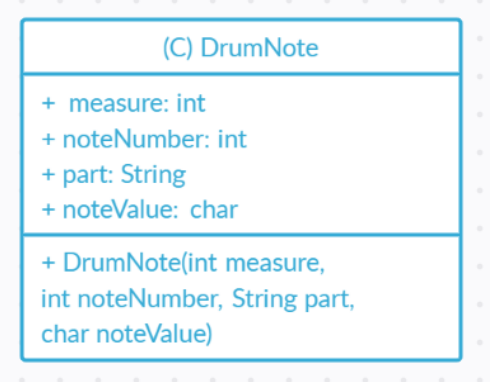
3.4.1. Description

When a drum note from the tablature is recognized, this class stores all the useful data the MusicXML file converter will need.

3.4.2. Methods

public DrumNote(int measure, int noteNumber, String part, char noteValue)
The default constructor from the DrumNote Class. It takes in 4 variables.

3.4.3. Class diagram



3.5. GuiWelcome

3.5.1. Description

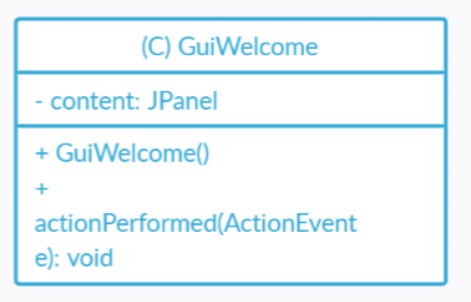
The welcome screen, along with making all of its functions connect to other classes.

3.5.2. Methods

public GuiWelcome()
Default constructor

public void actionPerformed(ActionEvent e)
Enter the button, once hit, the user will be presented with the next window.

3.5.3. Class diagram



3.6. GuiUploadWindow

3.6.1. Description

The class that puts together the upload window (Or tablature pasting area) works, along with making all of its functions connect to other classes.

3.6.2. Methods

public GuiUploadWindow(String title, String composer, String content)
Default constructor, takes the title of the tab, composer of the tab, and tab contents

public void actionPerformed(ActionEvent e) (Browse Button)
Event listener for the browse button, allows the user to browse for a file, then calls fileChecker to see if the file is valid. Calls error screen if file is not valid.

public void actionPerformed(ActionEvent e) (Convert Button)
Event listener for convert button, calls start() inside main to start the conversion process.

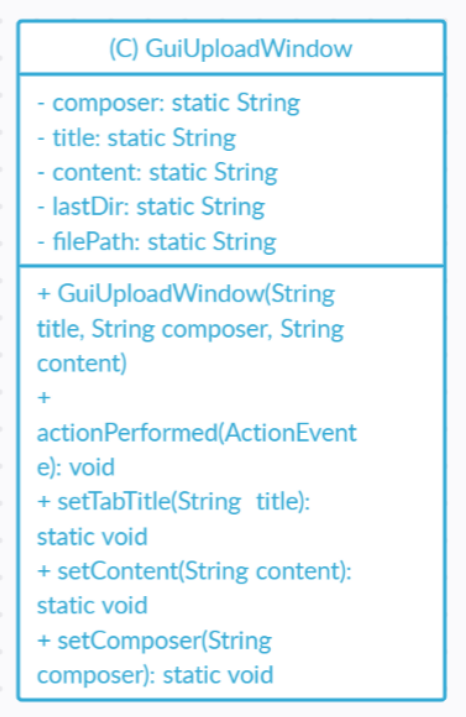
public void actionPerformed(ActionEvent e) (Modifications Button)
Event listener for modifications button. Will open the modifications menu for the user to put in modifications to the tab

public static void setTabTitle(String title)
Sets the title of the tablature the user inputs, to be used in the file conversion.

public static void setContent(String content)
Sets the content of the tablature the user inputs, to be used in the file conversion.

public static void setComposer(String composer)
Sets the composer of the tablature the user inputs, to be used in the file conversion.

3.6.3. Class diagram



3.7. ModificationsPage

3.7.1. Description

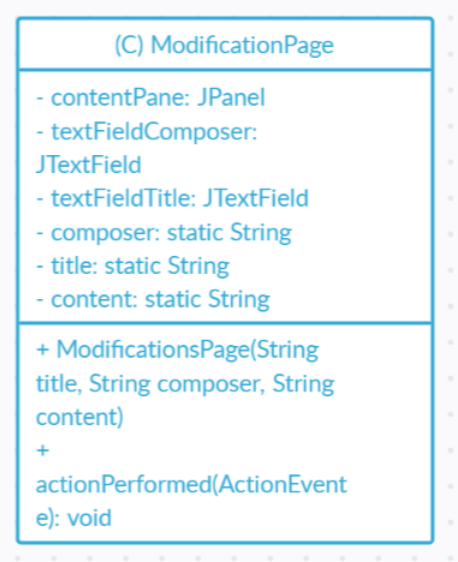
The class that puts together the modifications window works, along with making all of its functions connect to other classes.

3.7.2. Methods

public ModificationsPage(String title, String composer, String content)
Default constructor takes the title, composer, and content, so that it can be passed back to the upload window when returning back to preserve the information.

public void actionPerformed(ActionEvent e) (Confirm button)
Returns the user back to the upload window and saves the modifications that the user requires.

3.7.3. Class diagram



3.8. SaveFile

3.8.1. Description

The class that puts together the downloading the XML file window works, along with making all of its functions connect to other classes.

3.8.2. Methods

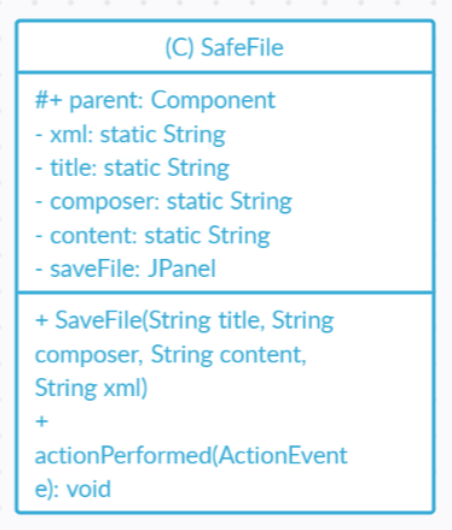
public SaveFile(String title, String composer, String content, String xml)
Default constructor, takes the title of the XML file, composer of the file , content of the file and the file itself.

public void actionPerformed(ActionEvent e) (Download button)
Event listener for Download button, saves the file and opens it in the users choice of application.

public void actionPerformed(ActionEvent arg0) (Exit button)
Event listener for Exit button. Closes the window and the program.

public void actionPerformed(ActionEvent e) (Edit button)
Event listener for Edit button. Goes back to the upload window to edit.

3.8.3. Class diagram



3.9. Error

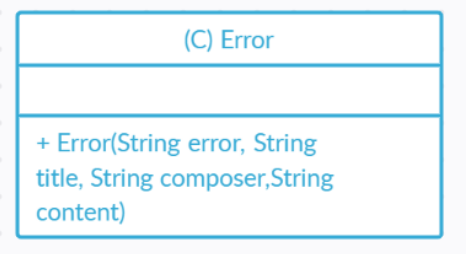
3.9.1. Description

Error screen that displays whenever an error occurs in the program.

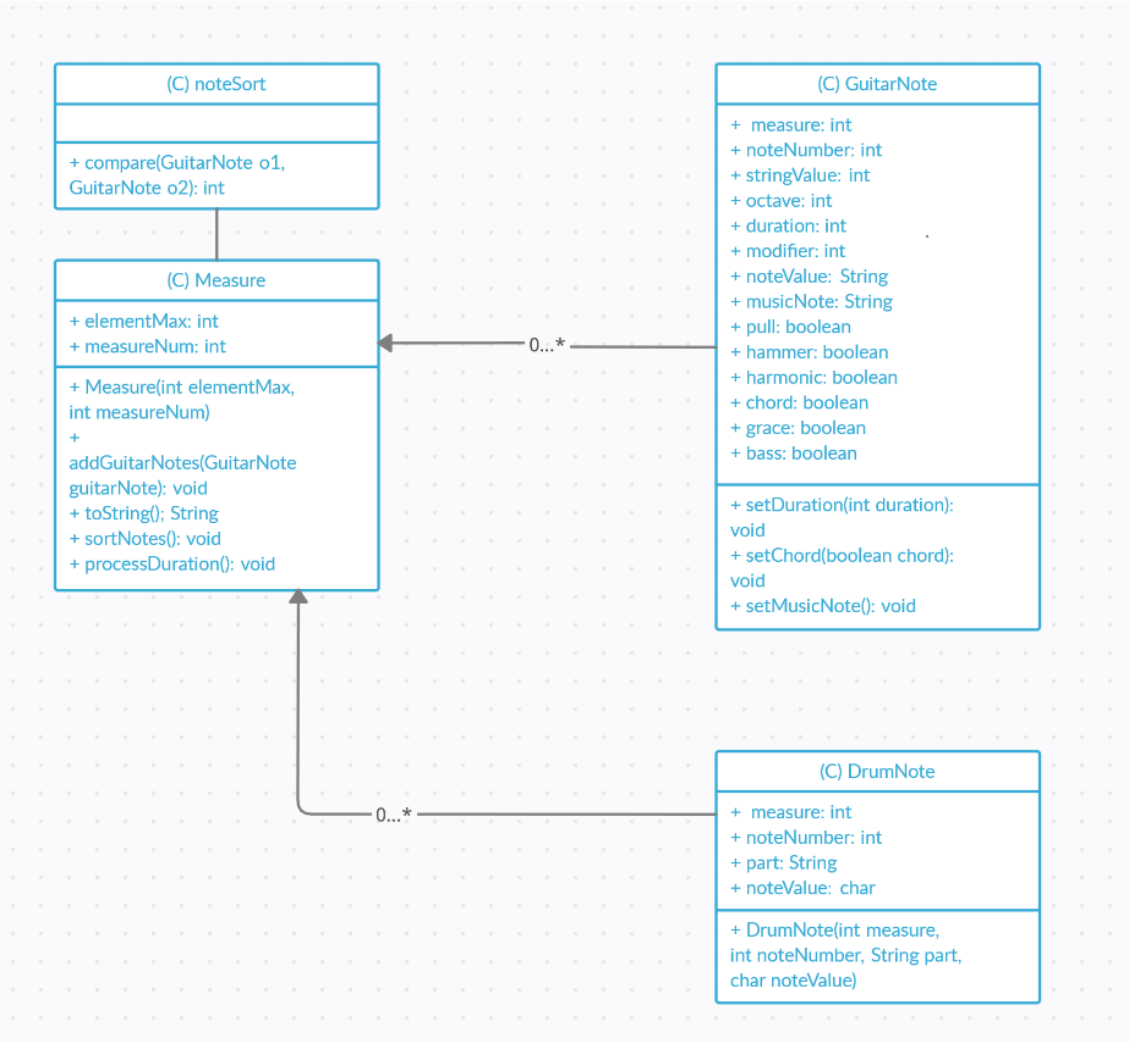
3.9.2. Methods

public Error(String error, String title, String composer, String content)
Default constructor takes the error message to display to the user, as well as the contents of the tab to preserve when returning back to the previous window.

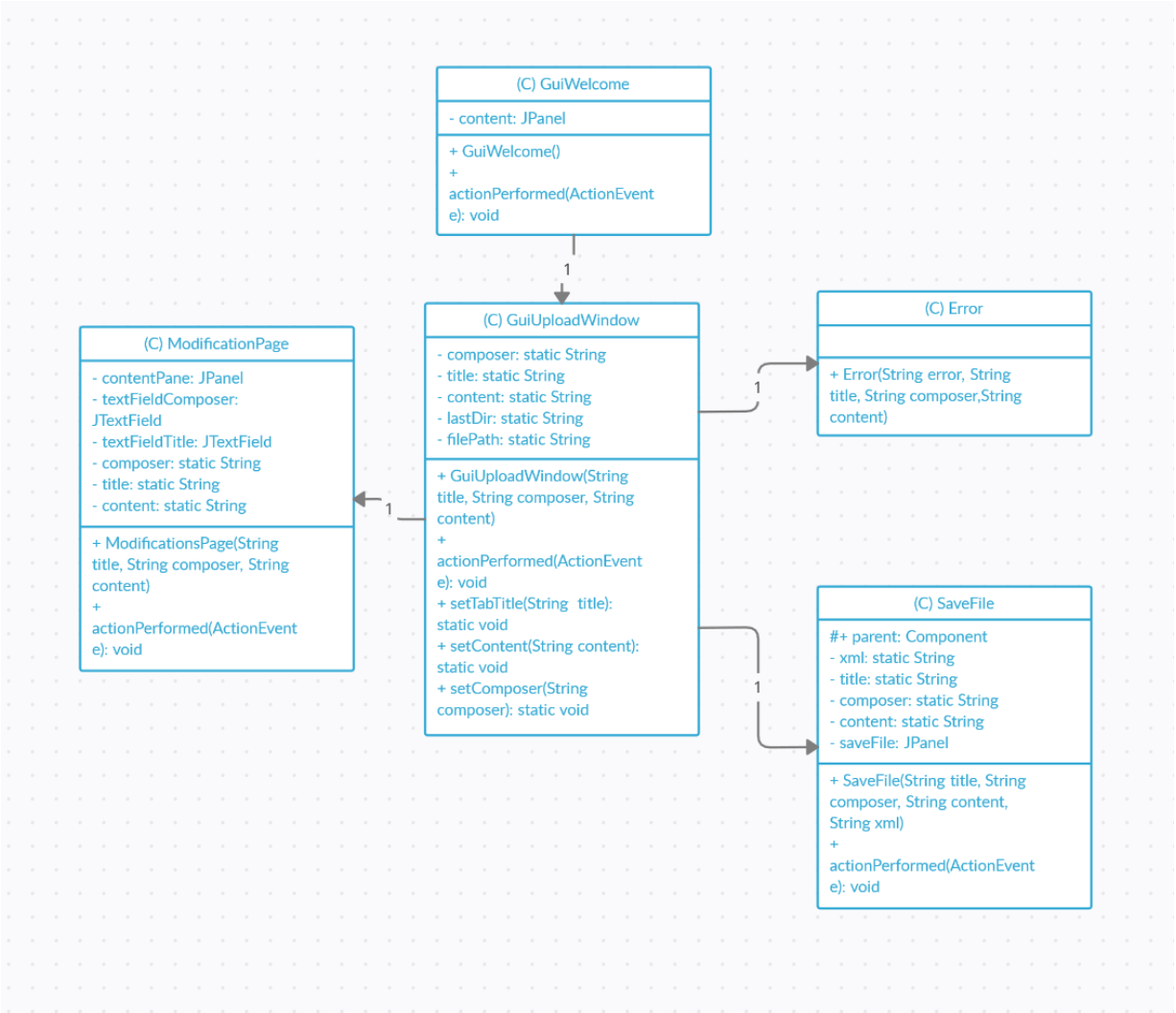
3.9.3. Class diagram



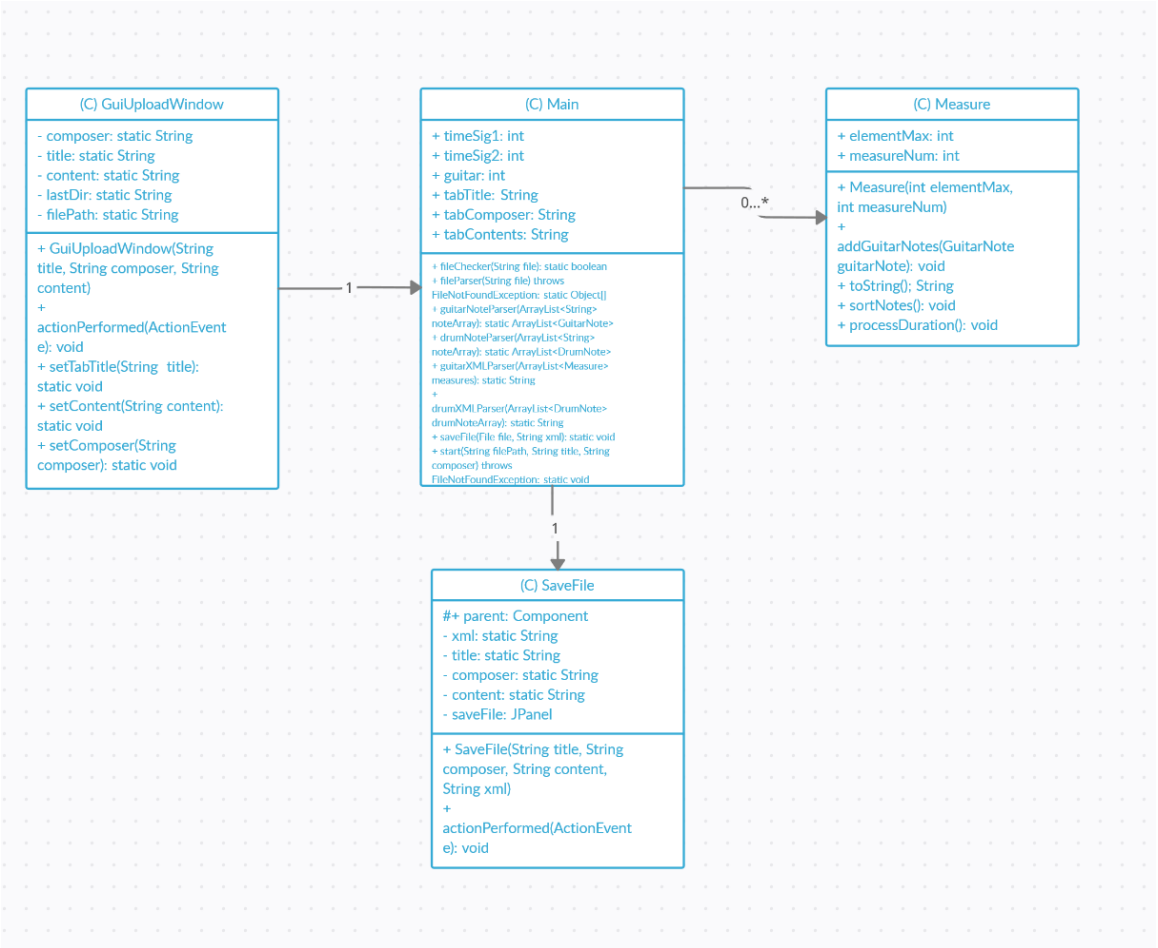
4. Software Overview



This represents the notes system of our program. Each measure is associated with either a drum note or guitar note. Since each measure contains an array of guitar or drum notes



This represents the GUI system. The welcome window calls the upload window, which then calls upon all the other related windows. Exactly one of each class may be called at a time.



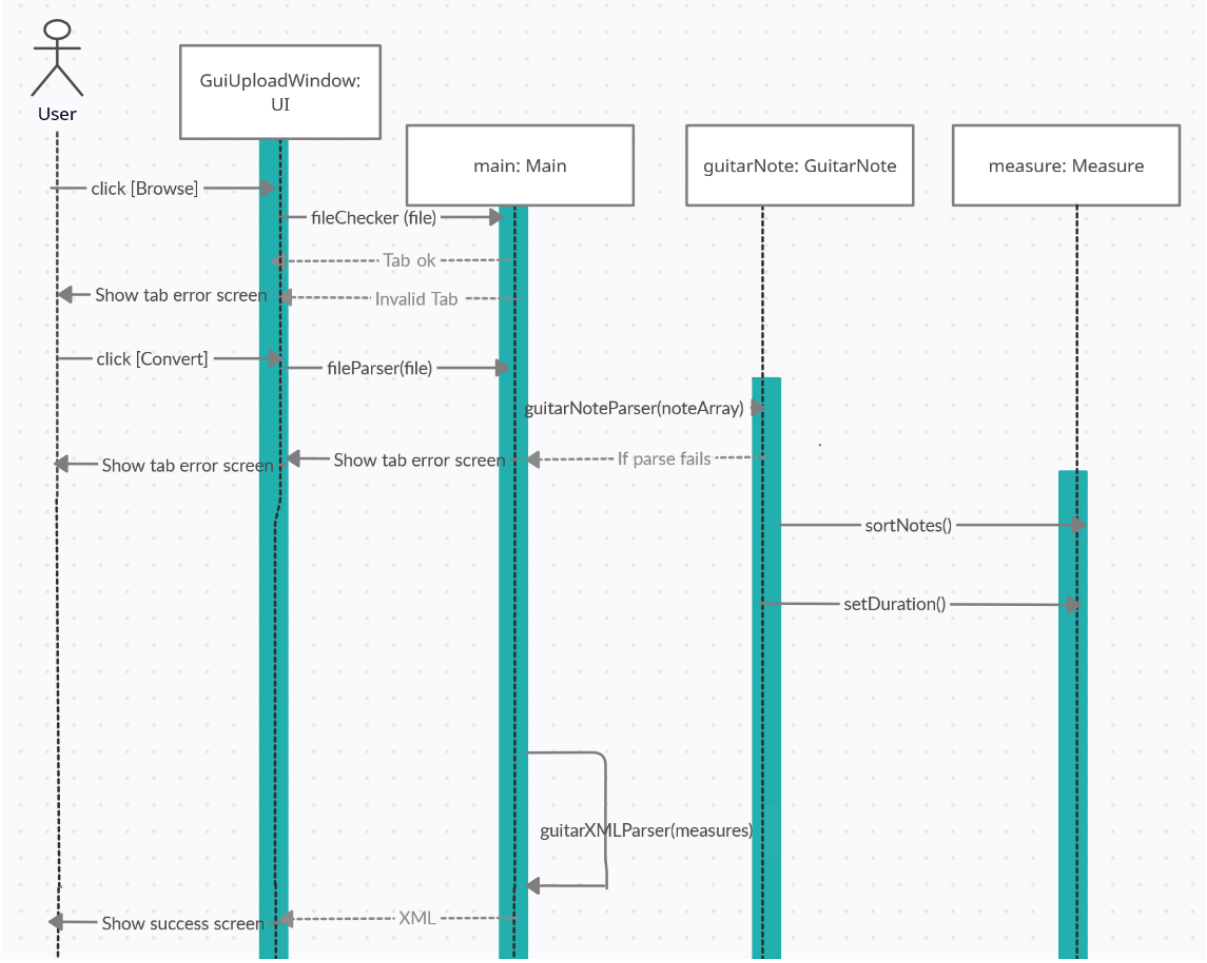
Overview of the main system. The GUI calls the main function which processes the tab, which is then saved to the user system.

5. Sequence Diagrams

Description: There are two main scenarios within the program, either the user chooses to paste in a tab, or the user chooses to browse for a tab.

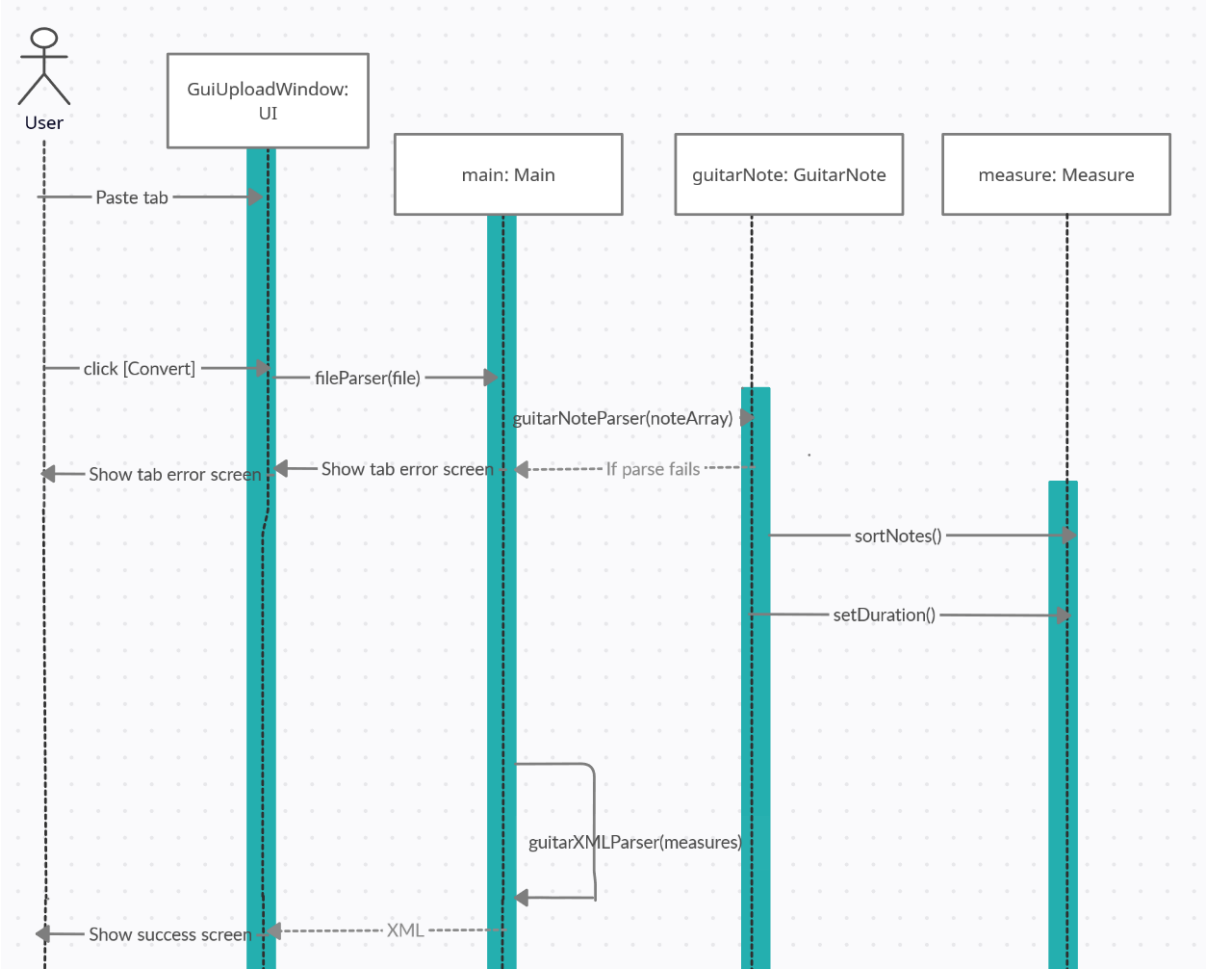
5.1. Scenarios

If a user were to browse a tab:



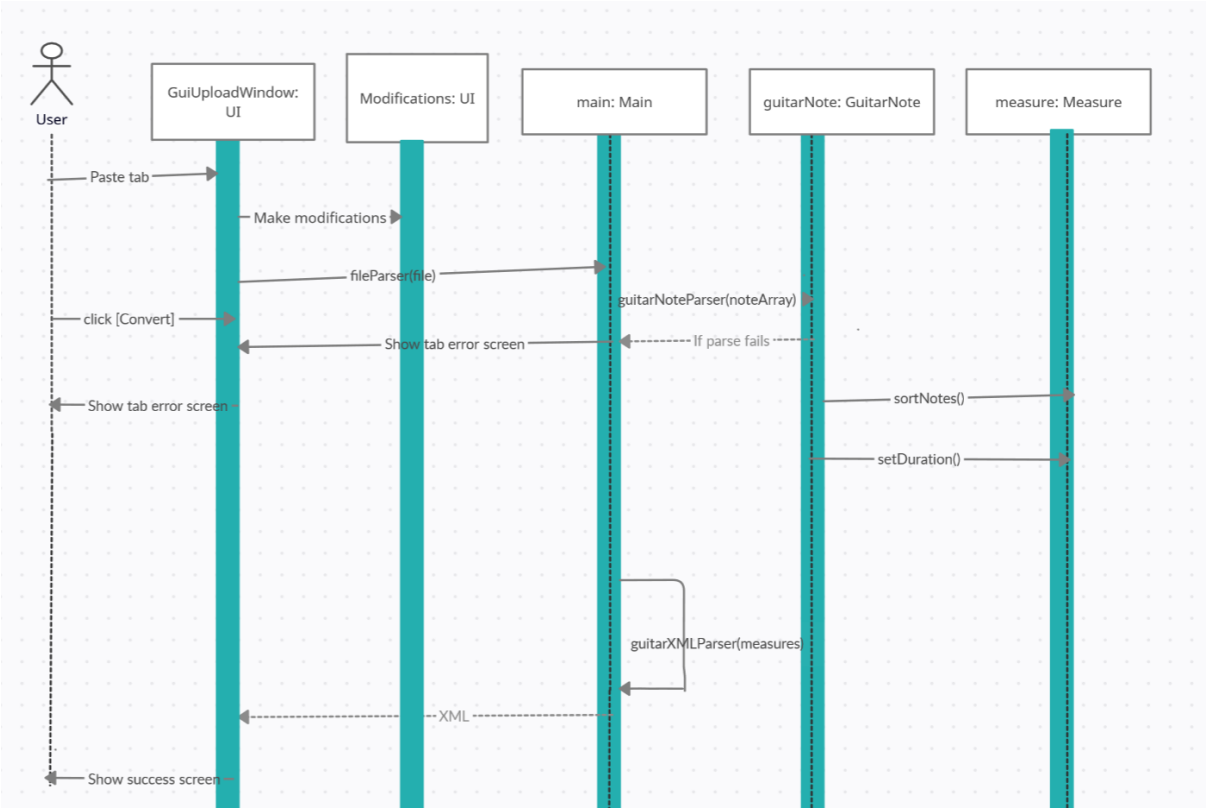
In this case, the upload window will call upon the browse window, which then checks the file type before proceeding.

If a user were to paste a tab:



In this case, the file checker is skipped and the program proceeds in it’s most vanilla form.

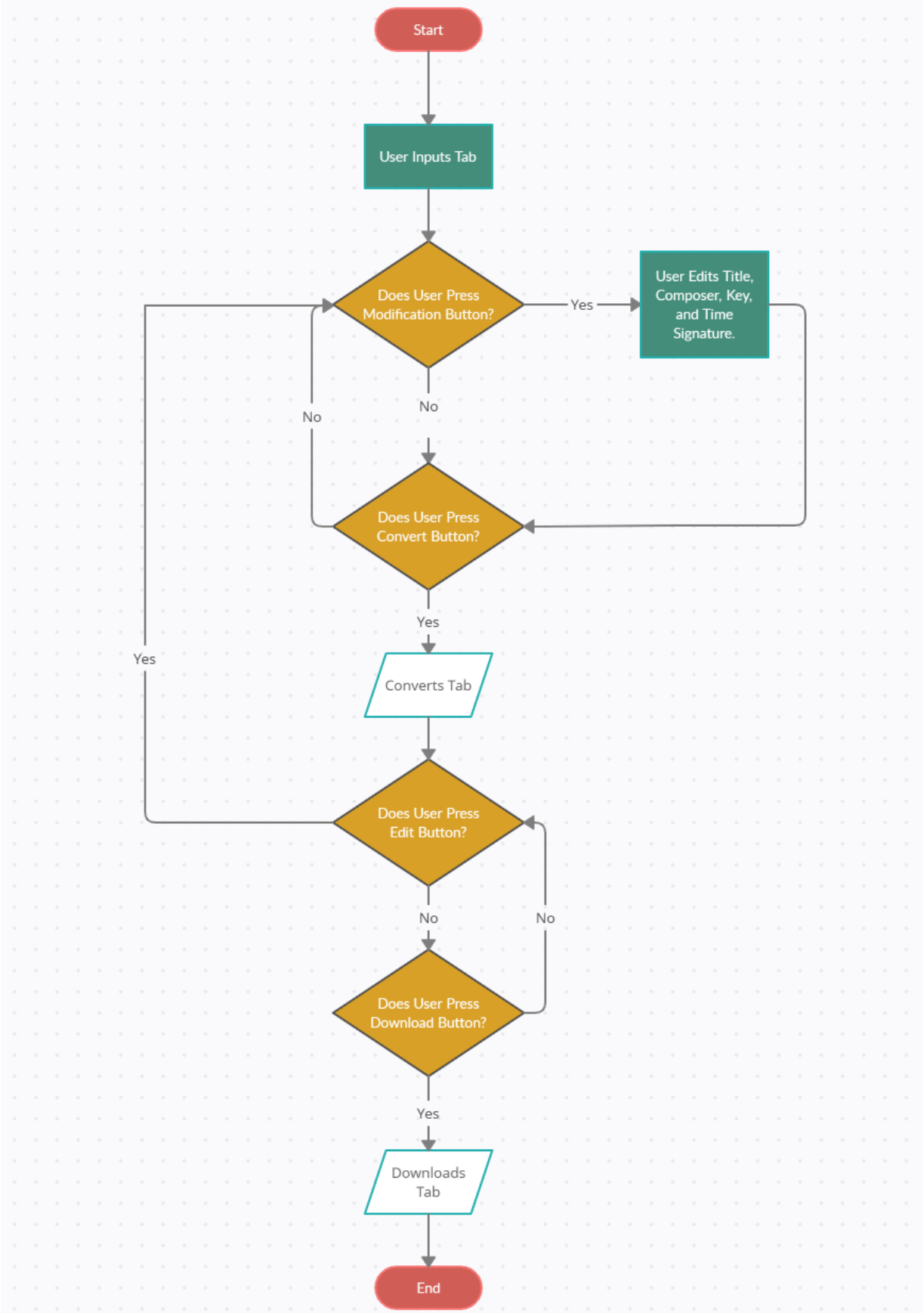
If a user were to make modifications to a tab:



In this case, a modification window UI is called from the upload window for the user to input any modification before proceeding with the program.

These sequence diagrams are sufficient as they cover the main runtime scenarios of the program. For inputs of drums, the sequence would remain exactly the same except for the different class/function names that are called.

6. Activity Diagram



7. Maintenance Scenarios

7.1. Adding new drum instruments

If we were to add support for drum instruments, like the cow bell for example, the *“public static String drumXMLParser(ArrayList<DrumNote> drumNoteArray)”* would need to be altered.

Within this function, there are if statements that check the instrument, or *“part”* of a given note. In a best case senator, adding a new instrument is as simple as adding another if statement and checks for this new instrument. However, sound instruments like a Pedal Hi-Hat would require extra steps, but there aren’t many of these.

7.2. Adding extra guitar strings

If support were to be added for extra guitar strings, the guitarNoteParser function will have to be modified. Since it currently checks for either 4 or 6 strings depending on the user selected instrument. If extra strings were to be added, this checker would have to be updated to support these new string amounts.

7.3. Adding modifiers to be recognized

If any modifiers were to be added, such as vibratos, or any other modifiers that are in the tabs, the guitarNoteParser function would have to be modified to recognize these extra modifiers. Currently, the function only looks for harmonics, hammer ons, or pull offs.

7.4. Adding extra instruments

If support for extra instruments were to be added, new parsers, note classes, and XML writers would have to be added. Currently, start() inside main calls upon the parser, then calls upon the XML writers which are specific for each instrument. Additionally, all the note classes are unique based on the instrument so new classes for the new instruments would have to be added.

7.5. Adding any changes to the XML

If any changes or additions were to be made to the XML, the XML parser function would have to be changed. There are separate XML parser functions for each instrument, so each one would have to be changed if the changes to the XML were universal. The XML parser is the only function that interacts with the output XML.

```
962             //process note here later
963             directives
964                     .add("note")
965                     .add("unpitched")
966                     .add("display-step");
967                     if (drumNote.part == "SD" || drumNote.part == "S") { //Snare
968                         directives.set("C");
969                     } else if (drumNote.part.equalsIgnoreCase("BD") || drumNote.part.equalsIgnoreCase("B")) { //Bass
970                         directives.set("F");
971                     } else if (drumNote.part.equalsIgnoreCase("HH") || drumNote.part.equalsIgnoreCase("H")) { //High hat
972                         directives.set("G");
973                     } else if (drumNote.part.equalsIgnoreCase("RD") || drumNote.part.equalsIgnoreCase("R")) { //Ride
974                         directives.set("F");
975                     } else if (drumNote.part.equalsIgnoreCase("CR") || drumNote.part.equalsIgnoreCase("C")) { //Crash
976                         directives.set("A");
977                     } else if (drumNote.part.equalsIgnoreCase("ST") || drumNote.part.equalsIgnoreCase("T1")) { //High Tom
978                         directives.set("E");
979                     } else if (drumNote.part.equalsIgnoreCase("MT") || drumNote.part.equalsIgnoreCase("T2")) { //Medium Tom
980                         directives.set("D");
981                     } else if (drumNote.part.equalsIgnoreCase("FT") || drumNote.part.equalsIgnoreCase("T3")) { //Floor Tom
982                         directives.set("A");
983                     } else {
984                         directives.set("F");
985                     }
```

7.6. Updating the logo

The logo would have to be updated through the GUI, and uploaded in a .png format using the setIcon function built into SWT WindowsBuilder.

7.7. Updating the GUI

Updates to the GUI in terms of new input buttons and parameters were to be added; they would be added using SWT WindowsBuilder through addition of methods. Each added functionality would have to have an actionListener that would have a specific condition to provide a reaction once an action is performed from the user's end. Also, the GUI would be stitched to the back end based on the specific task performed by the user.