Group 3

Testing Document

MusicXML Converter

Mohamed Ahmed 216542839
Maksim Kolotev 216509812
Mike Shen 216353583
Phuong Tran 215513666
Aman Patel 216823098

1. Introduction

The testing strategy for the application was to examine the logical functionality of the application.

1.1 Scope

1.1.1.1 In Scope

Logical aspects of the application will be tested. Changes will be made accordingly.

1.1.1.2 Out of Scope

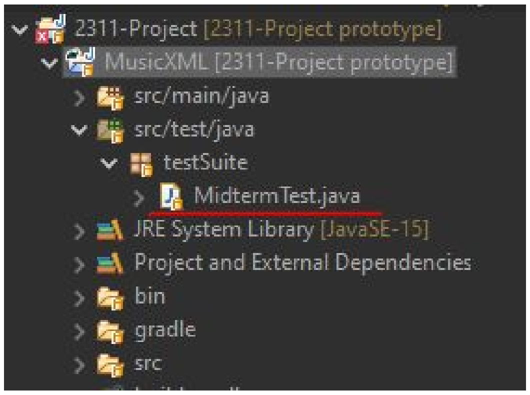
Event driven testing and testing the security of the application.

1.2 Quality Objective

The objective of the application is to create a universal format for common Western music notation, similar to the role that the MP3 format serves for recorded music. The musical information is designed to be usable by notation programs, sequencers and other performance programs, music education programs, and music databases. This software converts musical tabs into a MusicXML file. So, users can create musical notations out of the MusicXML file.

Some objectives of your testing project could be

- Test the musicXML file and test them in website for the validation: https://opensheetmusicdisplay.github.io/demo/
- Check to see if the correct information is being derived from the input text file (such as the correct key or number of measures in the text file)
- We can access the JUnit Test cases in the testSuite package within src/test/java (it must be within the src/test/java file directory in order to run the tests when building the gradle project).


2311-Project [2311-Project prototype]
MusicXML [2311-Project prototype]
src/main/java
src/test/java
testSuite
MidtermTest.java
JRE System Library [JavaSE-15]
Project and External Dependencies
bin
gradle
src

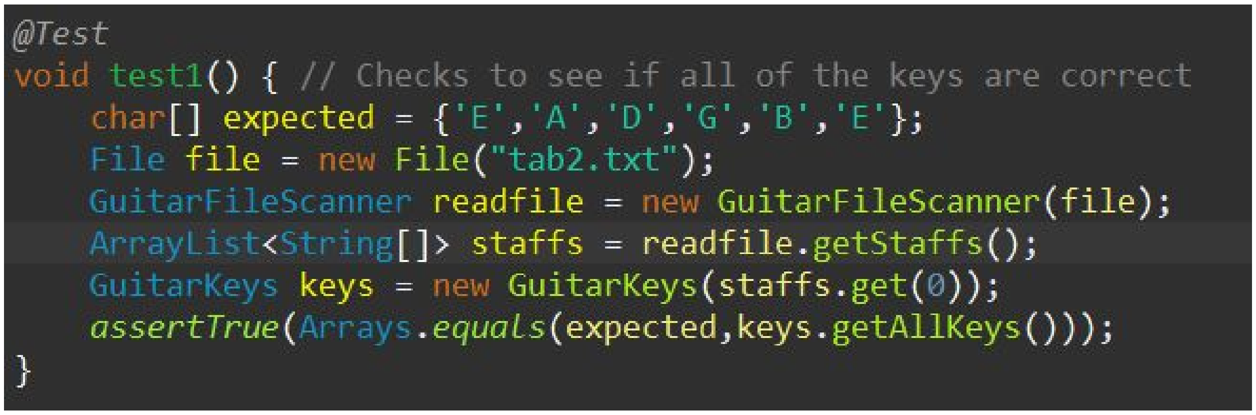
2. Test Methodology

2.1 Overview

The logical aspect of the application will be the focus for the testing. Upon exploration of the assignment requirement, it was discovered that it is not required to test the GUI of the application. For all the tests, We are using the tab2.txt file (File contents displayed at the end).

2.2 Test Deliverables

<u>Test Case 1:</u> testKeys: Check to see if all of the keys are present or not.



```java
@Test
void test1() { // Checks to see if all of the keys are correct
    char[] expected = {'E','A','D','G','B','E'};
    File file = new File("tab2.txt");
    GuitarFileScanner readfile = new GuitarFileScanner(file);
    ArrayList<String[]> staffs = readfile.getStaffs();
    GuitarKeys keys = new GuitarKeys(staffs.get(0));
    assertTrue(Arrays.equals(expected,keys.getAllKeys()));
}
```

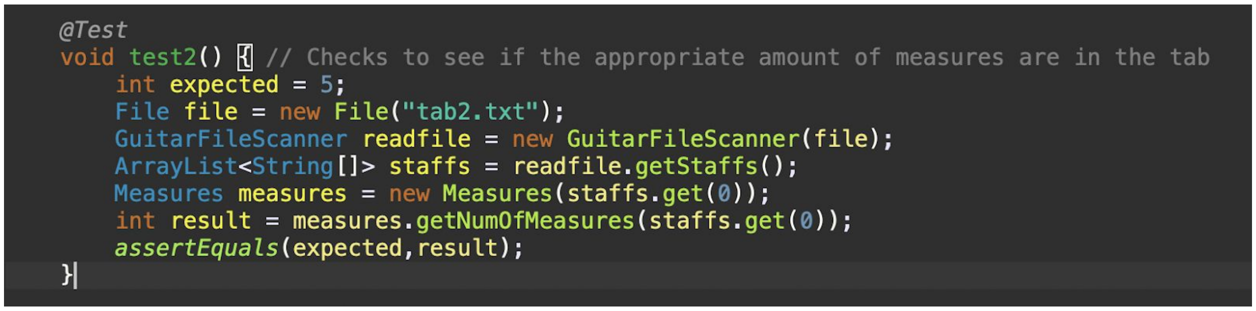
<u>Expected</u>:

- All the keys present in the tab2.txt file. For example: {‘E’ ,’A’, ‘D’, ‘G’, ‘B’, ‘E’}
- Standard tuning is assumed

<u>Actual:</u>

- Returns true. If all the expected keys are present in file/Tabs.
- Uses the getAllKeys method in GuitarKeys class

<u>Test Case 2:</u> measureTab: Checks to see if there is the appropriate amount of measures within each tab (Separated by a “|” character.



```java
@Test
void test2() { // Checks to see if the appropriate amount of measures are in the tab
    int expected = 5;
    File file = new File("tab2.txt");
    GuitarFileScanner readfile = new GuitarFileScanner(file);
    ArrayList<String[]> staffs = readfile.getStaffs();
    Measures measures = new Measures(staffs.get(0));
    int result = measures.getNumOfMeasures(staffs.get(0));
    assertEquals(expected,result);
}
```

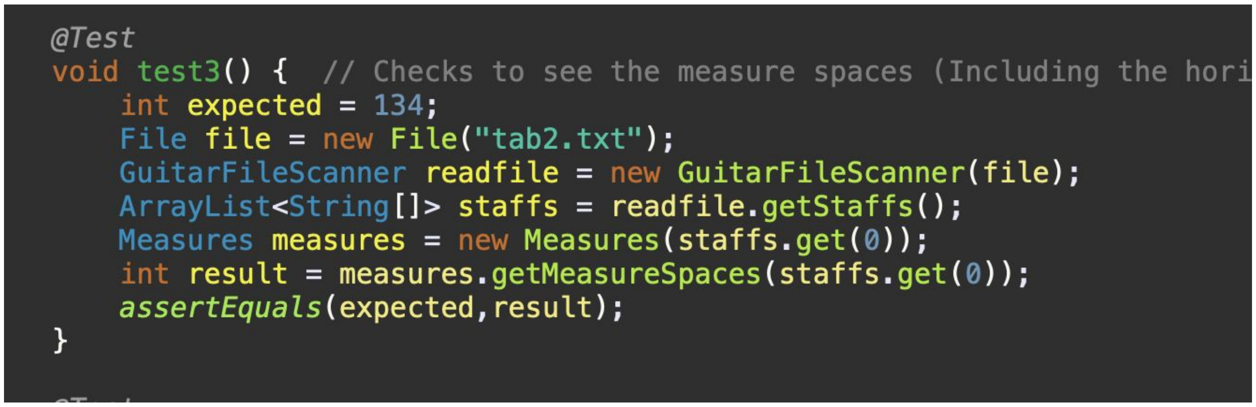
<u>Expected:</u>

- Integer, Number of measures in the tab.

Actual:

- Returns True if the amount of measures equal to the expected amount.
- Uses the getNumOfMeasures method in the Measures class

Test Case 3: testSpaceMeasure: Checks to see the measure spaces (Including the horizontal lines in between excluding the horizontal lines at the very beginning/end)



```
@Test
void test3() {  // Checks to see the measure spaces (Including the hori
    int expected = 134;
    File file = new File("tab2.txt");
    GuitarFileScanner readfile = new GuitarFileScanner(file);
    ArrayList<String[]> staffs = readfile.getStaffs();
    Measures measures = new Measures(staffs.get(0));
    int result = measures.getMeasureSpaces(staffs.get(0));
    assertEquals(expected,result);
}
```

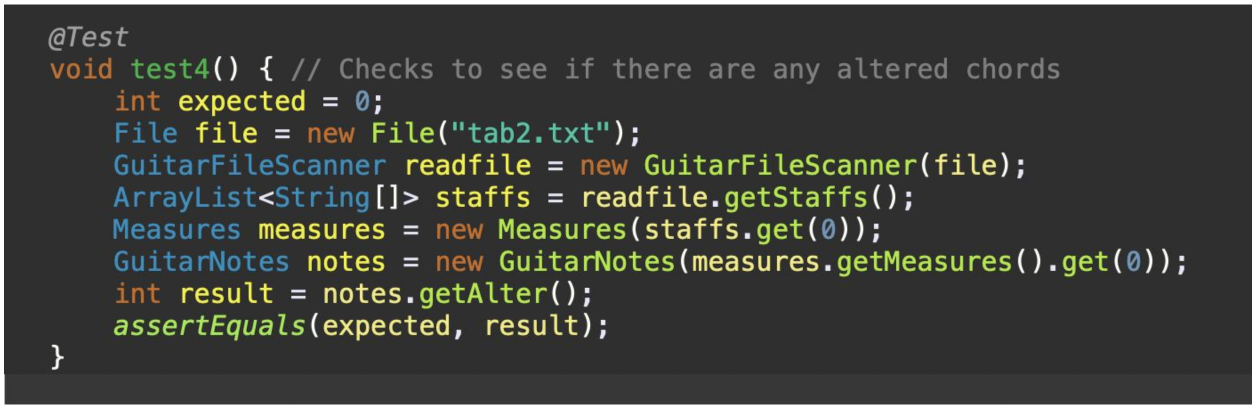
Expected:

- Integer (134 number of measures including the "|" characters that are in between measures excluding the first and last horizontal line character)

Actual:

- Returns True if the amount of measure spaces equal to the expected amount.
- Uses the getMeasureSpaces in the Measures class

Test Case 4: alteredChordTest: Checks to see if there are any altered chords



```
@Test
void test4() { // Checks to see if there are any altered chords
    int expected = 0;
    File file = new File("tab2.txt");
    GuitarFileScanner readfile = new GuitarFileScanner(file);
    ArrayList<String[]> staffs = readfile.getStaffs();
    Measures measures = new Measures(staffs.get(0));
    GuitarNotes notes = new GuitarNotes(measures.getMeasures().get(0));
    int result = notes.getAlter();
    assertEquals(expected, result);
}
```

Expected**:**

- Integer 0, For the assurance that no chords i.e. tabs were altered.

Actual**:**

- Returns false if any of the chords were altered in the textArea.
- Returns true if any of the chords were not altered or affected in the textArea.

2.3 Test Completeness

Here you define the criterias that will deem your testing complete.
For instance, a few criteria to check Test Completeness would be
- 100% test coverage
- All Manual & Automated Test cases executed
- All open bugs are fixed or will be fixed in next release

2.4 Test Environment

It mentions the minimum hardware requirements that will be used to test the Application.

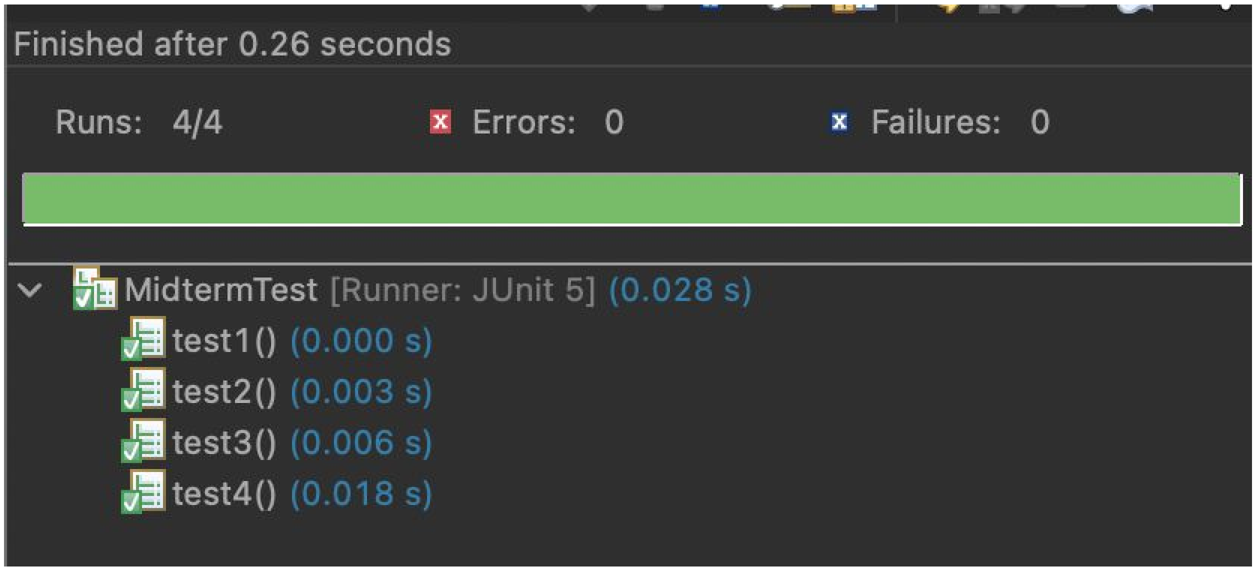
Following software's are required in addition to client-specific software.
- Windows 7 and above / Linux / MacOS
- Eclipse / IntelliJ
- JUnit 15 library
- Java SE 15 library

Conclusion

The test cases provided above test everything functional in the MusicXML Convertor. The test cases check the core functionality of how a MusicXML Convertor should work. Another thing tested is how the tabs are converted when you copy and paste them and compare with the current file open, To see if they are not the same. All in all, these test cases are sufficient in testing our tool for errors and bugs.

This is the example txt file we used for the JUnit testing. (tab2.txt)

```
|--0--------------1-------|-------------------------|--0----------------------|-------------------------|-------------------------|
|-----------------3-----5-|-2-----------------------|-----------------3-----5-|-2----------------3------|----0--------------------|
|-----------------3-------|-2-----------------------|-----------------3-----5-|-2----------------3------|-------------------------|
|-----------------5-------|-2-----------------------|-----------------3-----5-|-2----------------5------|-------------------------|
|-------------------------|-0-----------------------|-----------------3-----5-|-2-----------------------|-------------------------|
|-------------------------|-------------------------|-----------------3-----5-|-2-----------------------|-------------------------|
```


```
Finished after 0.26 seconds

Runs:  4/4        Errors:  0        Failures:  0

MidtermTest [Runner: JUnit 5] (0.028 s)
    test1() (0.000 s)
    test2() (0.003 s)
    test3() (0.006 s)
    test4() (0.018 s)
```