MusicXML - Testing Document

-By: Group 3 EECS 2311

Group 3:

Aman Patel
Phuong Tran
Mike Shen
Maksim Kolotev

Table Of Contents

Page No.

1. Introduction

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

1.1 Scope

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

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).

 # All the testing code could be found in the TestSuite and In the Midterm.java file. We are using Unit Testing for testing out our system.

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

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

```
@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()));
}
```

Expected:

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

Actual:

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

Test Case 2: measureTab: Checks to see if there is the appropriate amount of measures within each tab (Separated by a "|" character).

```
@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);
}
```

Expected:

- 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.

Test Case 5: InstrumentDetectionTest: Checks to see if the instrument detected upon pasting or viewing the tabs

```
    /*------------------------ Instrument Detection Test Case ------------------------*/

    @Test
    void test18() {
        File file = new File("bass.txt");
        InstrumentDetection detect = new InstrumentDetection(file);
        assertTrue(detect.getDetectedInstrument() != null);
    }
```

Expected:

- Bass, As the bass drum tabs were used for the testing purpose

Actual:

- Returns Bass if the amount of measure spaces equal to the expected amount.
- Uses the getDetectedInstrument in the Instrument Detection class

Test Case 6: CompleteBassTesting: Checks all Bass functions used in the system by inputting sample Bass tabs and console logging important information and converting it into XML format respectively.

```
197
198     /*---------------------------- Complete Bass Tabs Testing ----------------------------*/
199
200     @Test
201     void test16() {
202         File file = new File("sampleBass.txt");
203         BassFileScanner readFile = new BassFileScanner(file);
204         ArrayList<String[]> staffs = readFile.getStaffs();
205         BassKeys keys = new BassKeys(staffs.get(0));
206         keys.getAllKeys();
207         for (int i = 0; i < staffs.size(); i++) {
208             for (int j = 0; j < 4; j++) {
209                 System.out.println(staffs.get(i)[j]);
210             }
211         }
212         BassMeasures measures = new BassMeasures(staffs.get(0));
213         for (int i = 0; i < measures.getMeasures().size(); i++) {
214             for (int j = 0; j < 4; j++) {
215                 System.out.println(measures.getMeasures().get(i)[j]);
216             }
217         }
218         BassNotes notes = new BassNotes(measures.getMeasures().get(0));
219         Map<Pair<Integer, Integer>, List<Integer>> notesMap = notes.getNotesMapping();
220         for (Entry<Pair<Integer, Integer>, List<Integer>> entry : notesMap.entrySet()) {
221             Integer index = entry.getKey().getKey();
222             List<Integer> value = entry.getValue();
223             for (int i = 0; i < value.size(); i++) {
224                 value.get(i);
225                 notes.getOctave(entry.getKey().getValue(), value.get(i));
226             }
227         }
228         BassXMLOut test = new BassXMLOut();
229         assertTrue(test.convertToXML(file) != null);
230     }
231
```

Expected:

- Bass Tabs converted into musicXML file format and some important information in console

Actual:

- Returns the Bass musicXML file which we see if it's there or not by null checking
- Multiple Bass classes have been used in order to convert tab into musicXML

Test Case 7: CompleteDrumTesting: Checks all Drum functions used in the system by inputting sample Drum tabs and console logging important information and converting it into XML format respectively.

```
/*---------------------- Complete Drum Testing ----------------------*/

@Test
void test17() {
    File file = new File("sampleDrums.txt");
    DrumFileScanner readFile = new DrumFileScanner(file);
    ArrayList<List<String>> staffs = readFile.getDrumStaffs();
    for (int i = 0; i < staffs.size(); i++) {
        for (int j = 0; j < staffs.get(i).size(); j++) {
            System.out.println(staffs.get(i).get(j));
        }
        System.out.println();
    }
    DrumMeasures measures = new DrumMeasures(staffs.get(0));
    for (int i = 0; i < measures.getMeasures().size(); i++) {
        for (int j = 0; j < measures.getMeasures().get(i).size(); j++) {
            System.out.println(measures.getMeasures().get(i).get(j));
        }
        System.out.println();
    }
    DrumInstrument instruments = new DrumInstrument(staffs.get(1));
    for (int i = 0; i < instruments.getAllInstr().length; i++) {
        System.out.println(instruments.getAllInstr()[i]);
    }
    DrumNotes notes = new DrumNotes(measures.getMeasures().get(0));
    System.out.println();
    System.out.println("Testing constant [vertical.size()]: " + notes.vertical.size());
    System.out.println();
    Map<Pair<Integer, Integer>, List<Character>> notesMap = notes.getNotesMapping();
    for (Entry<Pair<Integer, Integer>, List<Character>> entry : notesMap.entrySet()) {
        Integer index = entry.getKey().getKey();
        Integer gString = entry.getKey().getValue();
        List<Character> value = entry.getValue();

        System.out.print("At index: " + index + " ");
        System.out.print("At string: " + gString + " ");
        System.out.print("   Values: ");
        for (int i = 0; i < value.size(); i++) {
            System.out.print(value.get(i) + " ");
        }
        System.out.println();
    }
    DrumDuration dur = new DrumDuration(notes.getNotesMapping(), measures.getMeasureSpaces(measures.getMeasures().get(0)));
    DrumDuration dur2 = new DrumDuration(notes.getNotesLowMapping(), measures.getMeasureSpaces(measures.getMeasures().get(0)));
    DrumXMLOut n = new DrumXMLOut();
    File out = n.convertToXML(file);
    assertTrue(dur != null && dur2 != null && out != null);
}
```

Expected:

- Drum Tabs converted into musicXML file format and some important information in console

Actual:

- Returns the Drum musicXML file which we see if it's there or not by null checking
- Multiple Drum classes have been used in order to convert tab into musicXML

Test Case 8: CompleteGuitarTesting: Checks all Guitar functions used in the system by inputting sample guitar tabs and console logging important information and converting it into XML format respectively.

```java
/*------------------------ Complete Guitar Test Case ----------------------------*/

@Test
void test19() {
    File file = new File("sampleGuitar.txt");
    GuitarFileScanner readFile = new GuitarFileScanner(file);

    ArrayList<String[]> staffs = readFile.getStaffs();
    GuitarKeys keys = new GuitarKeys(staffs.get(0));

    keys.getAllKeys();

    for (int i = 0; i < staffs.size(); i++) {
        for (int j = 0; j < 6; j++) {
            System.out.println(staffs.get(i)[j]);
        }
        System.out.println();
    }

    GuitarMeasures measures = new GuitarMeasures(staffs.get(0));

    for (int i = 0; i < measures.getMeasures().size(); i++) {
        for (int j = 0; j < 6; j++) {
            System.out.println(measures.getMeasures().get(i)[j]);
        }
        System.out.println();
    }

    GuitarNotes notes = new GuitarNotes(measures.getMeasures().get(0));

    Map<Pair<Integer, Integer>, List<Integer>> notesMap = notes.getNotesMapping();

    for (Map.Entry<Pair<Integer, Integer>, List<Integer>> entry : notesMap.entrySet()) {
        Integer index = entry.getKey().getKey();
        Integer gString = entry.getKey().getValue();
        List<Integer> value = entry.getValue();

        System.out.print("At index: " + index + " ");
        System.out.print("At string: " + gString + " ");
        System.out.print("   Values: ");
        for (int i = 0; i < value.size(); i++) {
            System.out.print(value.get(i) + " ");
        }
        System.out.println();
    }

    GuitarXMLOut test = new GuitarXMLOut();
    assertTrue(test.convertToXML(file) != null);
}
```

Expected:

- Guitar Tabs converted into musicXML file format and some important information in console

Actual:

- Returns the Guitar musicXML file which we see if it's there or not by null checking
- Multiple Guitar classes have been used in order to convert tab into musicXML

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

- 70% test coverage
- All Manual & Automated Test cases executed
- The Rest 30% of Testing is left is of the GUI Testing

MusicXML	72.8 %	9,242	3,445	12,687
src/main/java	70.3 %	8,042	3,396	11,438
MusicXML	70.3 %	8,042	3,396	11,438
BassDuration.java	93.2 %	234	17	251
BassFileScanner.java	88.9 %	185	23	208
BassKeys.java	100.0 %	36	0	36
BassMeasures.java	96.7 %	118	4	122
BassNotes.java	38.6 %	550	874	1,424
BassTuning.java	95.5 %	63	3	66
BassXMLOut.java	93.0 %	822	62	884
DrumDuration.java	93.4 %	282	20	302
DrumFileScanner.java	95.3 %	122	6	128
DrumInstrument.java	100.0 %	81	0	81
DrumMeasures.java	97.0 %	130	4	134
DrumNotes.java	94.1 %	428	27	455
Drumset.java	100.0 %	337	0	337
DrumVoice.java	72.0 %	18	7	25
DrumXMLOut.java	97.9 %	2,091	44	2,135
GuitarDuration.java	94.2 %	226	14	240
GuitarFileScanner.java	94.4 %	238	14	252
GuitarKeys.java	100.0 %	36	0	36
GuitarMeasures.java	96.7 %	118	4	122
GuitarNotes.java	53.4 %	848	740	1,588
GuitarTuning.java	96.5 %	83	3	86
GuitarXMLOut.java	99.2 %	877	7	884
InstrumentDetection.java	63.0 %	119	70	189
Launcher.java	0.0 %	0	96	96
LauncherController.java	0.0 %	0	81	81
MainApp.java	0.0 %	0	31	31
MainController.java	0.0 %	0	637	637
Notes.java	0.0 %	0	605	605
NotesFinder.java	0.0 %	0	3	3
src/test/java	96.1 %	1,200	49	1,249

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 10 / MacOS
- Eclipse
- 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.

Reason For Leaving GUI Test:

GUI test automation is insanely hard, as proven by the sad test execution metrics from around the globe. These three mistakes are behind those failures. Make even one of them and your test automation project is guaranteed to fail. Make all three and even God himself won't be able to untangle the mess. This is what takes up the rest of the 30% that can not be tested.

Reason Why ~70% is Enough:

Group 3's MusicXML converter consists of two different main parts. The first part being all the musical instruments classes which make up and build the XML files the user will see when converting. The second part is the GUI itself, which is the visual aspect of the program and what the user uses/navigates. The ratio between these two main parts is roughly 70/30. By the test coverage we can see that ~70% is covered, this means that all the backend code has been tested. Since the backend of the code is what actually does all the work this testing should be sufficient as it will test the functionality of the program and how it takes and processes given params. The other ~30% that does not get covered is the GUI section of the code. This does not need to be covered because of the reasons stated above ("Reason For Leaving GUI Test"), for the fact that it has been tested manually by our team, and that the GUI does not actually affect the functionality of the software.

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----------------------|------------------------|
```

sampleBass.txt

```
A major
G|-----------------|
D|-----------4-6-7-|
A|-----4-5-7-------|
E|-5-7-------------|
A minor
G|-----------------|
D|-------------5-7-|
A|-------5-7-8-----|
E|-5-7-8-----------|
```

sampleGuitar.txt

```
|----------0-----|-0--------------|
|---------0---0---|-0--------------|
|-------g0h1----1-|-1--------------|
|-----2-----------|-2--------------|
|---2-------------|-2--------------|
|-0---------------|-0--------------|
```

sampleDrum.txt

```
CC|x---------------|--------x-------|
HH|--x-x-x-x-x-x-x-|----------------|
SD|----o-------o---|oooo------------|
HT|----------------|----oo----------|
MT|----------------|------oo--------|
BD|o-------o-------|o-------o-------|

C |xx--------------|----------------|
HH|----x-x-x-x-x-x-|x-x-x-x-xox-x-x-|
T |----------------|----------------|
SD|----------------|----------------|
B |oo--------------|----------------|
```