

# **Test Document**

**EECS 2311 Group 1**

**Abdelrahman Altamimi**

**Hieu Le**

**Mahdiar Shoraka**

**Prabjot Dhaliwal**

**Yongjie Ba**

*Table of contents*

<b>1. Introduction .....</b>	<b>3</b>
<b>2. JavaFx GUI Test.....</b>	<b>3</b>
<b>2.1 GuiTest.java.....</b>	<b>4</b>
<b>3. JunitTest.....</b>	<b>11</b>
<b>3.1 ClefTest.java.....</b>	<b>11</b>
<b>3.2 InstrumentTest.java .....</b>	<b>13</b>
<b>3.3 MeasureTest.java.....</b>	<b>17</b>
<b>3.4 NotationTest.java .....</b>	<b>21</b>
<b>3.5 NoteTest.java .....</b>	<b>25</b>
<b>3.6 PartTest.java .....</b>	<b>30</b>
<b>3.7 ScoreTest.java .....</b>	<b>33</b>
<b>3.8 SlideTest.java .....</b>	<b>35</b>
<b>3.9 SlurTest.java .....</b>	<b>38</b>
<b>3.10 TiedTest.java .....</b>	<b>40</b>
<b>4. Model Tests .....</b>	<b>41</b>
<b>4.1 BeamingTest.java .....</b>	<b>41</b>
<b>4.2 PatternStringTest.java.....</b>	<b>43</b>
<b>5. Test Coverage Analysis.....</b>	<b>45</b>
<b>4.1 Test Summary.....</b>	<b>45</b>
<b>4.2 Test Coverage.....</b>	<b>45</b>

# 1 Introduction

This document describes JavaFX **GUI test cases** which can be found under the **src/test/java/GUI** folder and **Junit test cases** for most classes which can be found under the **src/test/java/Test\_Coverage** folder.

The GUI test cases are used to test the real-time functionality of button activations, creating new windows and interacting with the different screens in general.

The Test\_Coverage package is responsible for testing 3 sections of the program: the XML parsing, the audio playback, the GUI element generation for the previewer.

**XML Parsing Unit Tests:** Tests the parsing of the musicXML into organised java classes, so we can read and use the XML data in a much easier way. Test cases for each class are described in this document to ensure that all required information is extracted from the XML document and the information is correct

**Audio Playback Unit Tests:** Tests the creation of the JFugue Pattern string which is used to play the music using the JFugue library. Test cases ensure that the generated pattern string correctly represents the music to be played.

**GUI Element Generating Unit Tests:** Tests the creation of the graphical score using the output of MuseScore as a reference of correct output. This ensures that the output sheet music is correct. The elements tested are specifically those which are easily quantifiable without having to know the precise location on the page for these elements.

## 2 JavaFx GUI Test

### 2.1 GuiTest.java

This class is used to test the GUI. It uses a robot to click each button in order to test the GUI work correctly.

*Test Cases:*

**Name:** + start(Stage): void

**Description:** Used to load GUI

**Input:** Parent root =

```
FXMLLoader.load(getClass().getClassLoader().getResource("GUI/mainView.fxml"));
```

```
Scene scene = new Scene(root);
```

```
scene.getStylesheets().add(getClass().getClassLoader().getResource("GUI/styles.css").toExternalForm());
```

```
stage.setTitle("TAB 2 XML");
```

```
stage.setMinWidth(700);
```

```
stage.setMinHeight(500);
```

```
stage.setScene(scene);
```

```
stage.show();
```

**Output:** GUI

**Name:** + testEmptyInput(FxRobot)

**Description:** Testing in mainViewGUI by using empty input

**Input:** robot.clickOn("#previewButton");

```
FxAssert.verifyThat("#previewButton", NodeMatchers.isDisabled());
```

```
robot.clickOn("#showMXLButton");
```

```
FxAssert.verifyThat("#showMXLButton", NodeMatchers.isDisabled());
```

```
robot.clickOn("#saveMXLButton");
```

```
FxAssert.verifyThat("#saveMXLButton", NodeMatchers.isDisabled());
```

```
robot.clickOn("#saveTabButton");
```

```
FxAssert.verifyThat("#saveTabButton", NodeMatchers.isDisabled());
```

**Output:** True

**Name:** + invalidInput(FxRobot)

**Description:** Testing in mainViewGUI by using invalid input

**Input:** robot.clickOn("#mainText");  
 robot.write("this text is not a valid measure", 0);  
 FxAssert.verifyThat("#previewButton", NodeMatchers.isDisabled());  
**Output:** True

**Name:** + validInput1(FxRobot)  
**Description:** Testing previewButton by in mainViewGUI using valid input  
**Input:** robot.clickOn("#mainText");  
 robot.write(" 3/8\n"  
   + "|5-----|\n"  
   + "|--7---|\n"  
   + "|----6-|\n"  
   + "|-----|\n"  
   + "|-----|\n"  
   + "|-----|\n"  
   + "", 0);  
  
 robot.clickOn("#previewButton");  
 FxAssert.verifyThat("#previewButton", NodeMatchers.isEnabled());  
**Output:** True

**Name:** + validInput2(FxRobot)  
**Description:** Testing showMXLButton in mainViewGUI by using valid input  
**Input:** robot.clickOn("#mainText");  
 robot.write(" 3/8\n"  
   + "|5-----|\n"  
   + "|--7---|\n"  
   + "|----6-|\n"  
   + "|-----|\n"  
   + "|-----|\n"  
   + "|-----|\n"  
   + "", 0);  
  
 robot.clickOn("#showMXLButton");  
 FxAssert.verifyThat("#showMXLButton", NodeMatchers.isEnabled());  
**Output:** True

**Name:** + validInput3(FxRobot)  
**Description:** Testing saveMXLButton in mainViewGUI by using valid input  
**Input:** robot.clickOn("#mainText");  
 robot.write(" 3/8\n"  
   + "|5-----|\n"  
   + "|--7---|\n"  
   + "|----6-|\n"  
   + "|-----|\n"  
   + "|-----|\n"  
   + "|-----|\n"  
   + "", 0);

```
+ "|-----|\n"
+ "", 0);
```

```
robot.clickOn("#saveMXLButton");
FxAssert.verifyThat("#saveMXLButton", NodeMatchers.isEnabled());
```

**Output:** True

**Name:** + validInput4(FxRobot)

**Description:** Testing saveTabButton in mainViewGUI by using valid input

**Input:** robot.clickOn("#mainText");

```
robot.write(" 3/8\n"
+ "|5-----|\n"
+ "|--7---|\n"
+ "|----6-|\n"
+ "|-----|\n"
+ "|-----|\n"
+ "|-----|\n"
+ "", 0);
```

```
robot.clickOn("#saveTabButton");
FxAssert.verifyThat("#saveTabButton", NodeMatchers.isDisabled());
```

**Output:** True

**Name:** + TestPlay(FxRobot)

**Description:** Testing playButton in PreViewGUI by using valid input

**Input:** robot.clickOn("#mainText");

```
robot.write(" 3/8\n"
+ "|5-----|\n"
+ "|--7---|\n"
+ "|----6-|\n"
+ "|-----|\n"
+ "|-----|\n"
+ "|-----|\n"
+ "", 0);
```

```
robot.clickOn("#previewButton");
robot.clickOn("#playButton");
FxAssert.verifyThat("#playButton", NodeMatchers.isEnabled());
robot.sleep(500);
```

**Output:** True, preview GUI showed and music plays

**Name:** + TestPause(FxRobot)

**Description:** Testing pauseButton in PreViewGUI by using valid input

**Input:** robot.clickOn("#mainText");

```
robot.write(" 3/8\n"
```

```

+ "|5-----|\n"
+ "|--7---|\n"
+ "|----6-|\n"
+ "|-----|\n"
+ "|-----|\n"
+ "|-----|\n"
+ "", 0);

```

```

robot.clickOn("#previewButton");
robot.clickOn("#playButton");
robot.clickOn("#pauseButton");
FxAssert.verifyThat("#pauseButton", NodeMatchers.isEnabled());
robot.sleep(500);

```

**Output:** True and music paused

**Name:** + TestPlayAndPause(FxRobot)

**Description:** Testing switch playButton and pauseButton immediately in PreViewGUI by using valid input

**Input:** robot.clickOn("#mainText");

```

robot.write(" 3/8\n"
+ "|5-----|\n"
+ "|--7---|\n"
+ "|----6-|\n"
+ "|-----|\n"
+ "|-----|\n"
+ "|-----|\n"
+ "", 0);

```

```

robot.clickOn("#previewButton");
robot.clickOn("#playButton");
robot.clickOn("#pauseButton");
robot.clickOn("#playButton");
robot.clickOn("#pauseButton");
robot.clickOn("#playButton");
robot.clickOn("#pauseButton");
robot.clickOn("#playButton");
robot.clickOn("#pauseButton");
FxAssert.verifyThat("#pauseButton", NodeMatchers.isEnabled());
robot.sleep(500);

```

**Output:** True, and Play and Pause button works well

**Name:** +TestRepeat(FxRobot)

**Description:** Testing repeat symbol in PreViewGUI by using valid input

**Input:** robot.clickOn("#mainText");

```

robot.write(" 7/4r\n"

```

```

+ " REPEAT 8x\r\n"
+ "G---4-----|\r\n"
+ "D---4-----|\r\n"
+ "A-2---2-----2-5-|\r\n"
+ "E-----2-5-----|\r\n"
+ "\r\n"
+ "G-----|-----|---4-----|\r\n"
+ "D-4-4-4-----4-|-3---2-----|---4-----|\r\n"
+ "A-----4-4-4-|-2-----0---2---5-|-2-2-----2-5-|\r\n"
+ "E-----2-5-|-0---3-----|---2-5-----|", 0);
robot.clickOn("#previewButton");
robot.sleep(1000);

```

**Output:** True and Show correct repeat number and symbol in scrollpane

**Name:** +TestSetTempo(FxRobot)

**Description:** Testing set tempo and play in PreViewGUI by using valid input

```

Input: robot.clickOn("#mainText");
robot.write(" 7/4\r\n"
+ " REPEAT 8x\r\n"
+ "G---4-----|\r\n"
+ "D---4-----|\r\n"
+ "A-2---2-----2-5-|\r\n"
+ "E-----2-5-----|\r\n"
+ "\r\n"
+ "G-----|-----|---4-----|\r\n"
+ "D-4-4-4-----4-|-3---2-----|---4-----|\r\n"
+ "A-----4-4-4-|-2-----0---2---5-|-2-2-----2-5-|\r\n"
+ "E-----2-5-|-0---3-----|---2-5-----|", 0);
robot.clickOn("#previewButton");
robot.doubleClickOn("#tempoField");
robot.write("240",0);
robot.clickOn("#playButton");
robot.sleep(10000);

```

**Output:** True and music tempo changed to 240

**Name:** +TestGoToMeasure(FxRobot)

**Description:** Testing Go to measure feature in preview GUI by using valid input

```

Input: robot.clickOn("#mainText");
robot.write(" 3/8\r\n"
+ "|5-----|\n"
+ "|--7---|\n"
+ "|----6-|\n"
+ "|-----|\n"
+ "|-----|\n"

```



```
+ "|-----|\n"
+ "", 0);
```

```
robot.clickOn("#previewButton");
robot.doubleClickOn("#gotoMeasureField");
robot.write("1",0);
robot.clickOn("#goButton");
FxAssert.verifyThat("#goButton", NodeMatchers.isEnabled());
```

**Output:** True and show/ go to measure 1

**Name:** +TestCustomizeDisplay(FxRobot)

**Description:** Testing Customize Display GUI works well and choicebox can be choosed, reset/ apply/apply&close button works well

**Input:**robot.clickOn("#mainText");

```
robot.write(" 3/8\n"
+ "|5-----|\n"
+ "|--7---|\n"
+ "|----6-|\n"
+ "|-----|\n"
+ "|-----|\n"
+ "|-----|\n"
+ "", 0);
```

```
robot.clickOn("#previewButton");
robot.clickOn("#displayButton");
```

```
robot.clickOn("#fontValue");
robot.sleep(200);
robot.clickOn("#noteSpaceValue");
robot.sleep(200);
robot.clickOn("#lineSpaceValue");
robot.sleep(200);
robot.clickOn("#taleWidthValue");
robot.sleep(200);
robot.clickOn("#applyButton");
robot.sleep(200);
robot.clickOn("#resetButton");
robot.sleep(200);
robot.clickOn("#applyAndExitButton");
```

**Output:** True and show Customize Display GUI, buttons works well

**Name:** +TestExportPDF(FxRobot)

**Description:** Testing exportPDF button in preView GUI

**Input:**robot.clickOn("#mainText");

```
robot.write(" 3/8\n"  
  + "|5-----|\n"  
  + "|--7---|\n"  
  + "|----6-|\n"  
  + "|-----|\n"  
  + "|-----|\n"  
  + "|-----|\n"  
  + "\"", 0);
```

```
robot.clickOn("#previewButton");
```

```
robot.clickOn("#exportButton");
```

```
FxAssert.verifyThat("#exportButton", NodeMatchers.isEnabled());
```

**Output:** True and output save as window, input file name and choose file path, they pdf generated successfully

### 3.JunitTest

All following tests use the setUp() method in each test case class to generate a Score object with the provided file path, therefore setUp(String path) will create a Score containing the parsed musicXML data, which we can use to test the methods in the **custom\_component\_data** package with testing .musicxml files located in the **src/test/resources/system** directory.

**Input:** The file path to a .musicxml file as a String

**Testing Goal:** Matching the elements and the attributes of the .musicxml file to the data stored in the Score object (either directly or stored in objects within the Score object)

#### 3.1 ClefTest

**Clef** is specifically declared near the beginning of a measure, otherwise, it is the same as the previous measure. Contains 2 attributes: symbol (defined in <sign> </sign>) and line (defined in <line> </line>) Regarding the symbol, it is represented with letters from A to G. If the value mentioned in <sign> </sign> is either percussion or TAB, that means the symbol is G and the line number is 2.

##### Test cases:

**Name:** +clefConstructorTest1()

**Description:** the test checks if the class functions normally by creating a Clef object and checking if it is null

**Input:**

```
setUp("src/test/resources/system/demoDrumsSimple1.musicxml");
    Clef expected = new Clef('G', 2);
    Clef actual = score.getParts().get(0).getMeasures().get(0).getClef();
    assertNotNull(actual);
```

```
Assertions.assertTrue(expected.getSymbol() == actual.getSymbol(), "The
expected symbol of clef is " + expected.getLine() + "while the actual symbol is "
+ actual.getLine());
```

```
Assertions.assertTrue(expected.getLine() == actual.getLine(), "The expected line
of clef is " + expected.getLine() + "while the actual line is " + actual.getLine());
```

**Output:** True

**Name:** + clefTest1()

**Description:** This test uses demoDrumsSimple1.xml as the test file. The test checks if the retrieved info of the clef in the 1st measure of the song is correct or

not

**Input:**

```
setUp("src/test/resources/system/demoDrumsComplex1.musicxml");
    char expect = 'G';
char actual = score.getParts().get(0).getMeasures().get(0).getClef().getSymbol();
    Assertions.assertEquals(expect,actual);
```

**Output:** True

**Name:** + TestgetSymbol()

**Description:** Testing getSymbol() method from Clef.java

**Input:**

```
setUp("src/test/resources/system/demoDrumsComplex1.musicxml");
    char expect = 'G';
char actual = score.getParts().get(0).getMeasures().get(0).getClef().getSymbol();
```

**Output:** True

**Name:** + TestgetLine()

**Description:** Testing getLine() method in Clef.java

**Input:**

```
setUp("src/test/resources/system/demoDrumsComplex1.musicxml");
    int expect = 2;
    int actual =
score.getParts().get(0).getMeasures().get(0).getClef().getLine();
    Assertions.assertEquals(expect,actual);
```

**Output:** True

**Name:** + Testequals()

**Description:** Testing equals() method in Clef.java

**Input:**

```
setUp("src/test/resources/system/demoDrumsComplex1.musicxml");
    Clef clef1 = score.getParts().get(0).getMeasures().get(0).getClef();
    Clef clef2 = new Clef('G',2);
    Assertions.assertTrue(clef1.equals(clef2));
```

**Output:** True

### 3.2 InstrumentTest

**Instrument** is defined inside of <score-instrument> and <midi-instrument>. And Contains 7 attributes: id (defined in <score-instrument id = " " > or <midi-instrument id = " " >), name (defined in <instrument-name>), volume (defined in <volume>), pan (defined in <pan>), midiChannel (defined in <midiChannel>), midiProgram (defined in <midi-program>) and midipitched (defined in <midi-unpitched>)

#### Test cases:

**Name:** + toStringTest()

**Description:** Testing toString() method in Instrument.java.

**Input:** String expected = "{\n" + "\tid: " + "ABC123" + ",\n" + "\tname: " + "music1" + ",\n" + "\tmidiChannel: " + 1 + ",\n" + "\tmidiProgram: " + 2 + ",\n" + "\tmidiUnpitched: " + 3 + ",\n" + "\tvolume: " + 3.5 + ",\n" + "\tpan: " + 2.5 + ",\n" + "}";

```
Instrument in = new Instrument("ABC123","music1");
in.setMidiChannel(1);
in.setMidiProgram(2);
in.setMidiUnpitched(3);
in.setVolume(3.5);
in.setPan(2.5);
```

```
String actual = in.toString();
assertEquals(expected, actual);
```

**Output:** true

**Name:** + InstrumentTest1()

**Description:** This test uses demoBassSimple1.xml as the test file. The test's purpose is to check if there are any instruments played in the melody or not. Since there is no info regarding instrument in this file, it is expected that the method to retrieve the info will return null

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");

```
int numInstrument = 0;
for(int i = 0; i < score.getParts().get(0).getInstruments().size(); i++) {
    numInstrument++;
}
```

```

        assertEquals(13, numInstrument);
        Instrument expected1 = new Instrument("P1-I45", "Pedal Hi-Hat");
        Instrument actual1 =
score.getParts().get(0).getInstruments().get("P1-I45");
        Assertions.assertTrue(expected1.getId().equals(actual1.getId()));

```

```

        Assertions.assertTrue(expected1.getName().equals(actual1.getName()));

```

**Output:** True

**Name:** + InstrumentTest2()

**Description:** This test uses demoDrumsComplex1.xml as the test file. Since there are various instruments used to play the song, the test checks the total number of instruments then check to see if the retrieved info of an instrument is correct

**Input:** setUp("src/test/resources/system/demoGuitarSimple2.musicxml");

```

        int numInstrument = 0;
        for(int i = 0; i < score.getParts().get(0).getInstruments().size(); i++) {
            numInstrument++;
        }
        assertEquals(0, numInstrument);
        Instrument expected1 = null;
        Instrument actual1 = score.getParts().get(0).getInstruments().get("");
        Assertions.assertEquals(expected1, actual1);

```

**Output:** True

**Name:** +TestGetID()

**Description:** Test getID() method in Instrument.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");

```

        String actualID1 = score.getParts().get(0).getId();
        String expectID1 = "P1";
        Assertions.assertEquals(expectID1, actualID1);

        String actualID2 =
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getInstrumentID();
        String expectID2 = "P1-I50";
        Assertions.assertEquals(expectID2, actualID2);

```

**Output:** True

**Name:** + TestGetName()

**Description:** Test GetName() method in Instrument.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");

```

        String actualName1 = score.getParts().get(0).getName();
        String expectName1 = "Drumset";
        Assertions.assertEquals(expectName1, actualName1);

```

**Output:** True

**Name:** + TestgetMidiChannel()

**Description:** Test getMidiChannel() method in Instrument.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");

<pre> int actual = score.getParts().get(0).getInstruments().get("P1-I46").getMidiChannel(); int expect = 10; Assertions.assertEquals(expect, actual); </pre> <p><b>Output:</b> True</p>
<p><b>Name:</b> + TestgetMidiProgram()</p> <p><b>Description:</b> Test getMidiProgram() method in Instrument.class</p> <p><b>Input:</b> setUp("src/test/resources/system/demoDrumsComplex1.musicxml");</p> <pre> int actual = score.getParts().get(0).getInstruments().get("P1-I46").getMidiProgram(); int expect = 1; Assertions.assertEquals(expect, actual); </pre> <p><b>Output:</b> True</p>
<p><b>Name:</b> + TestgetMidiUnpitched()</p> <p><b>Description:</b> Test getMidiUnpitched() method in Instrument.class</p> <p><b>Input:</b> setUp("src/test/resources/system/demoDrumsComplex1.musicxml");</p> <pre> int actual = score.getParts().get(0).getInstruments().get("P1-I46").getMidiUnpitched(); int expect = 46; Assertions.assertEquals(expect, actual); </pre> <p><b>Output:</b> True</p>
<p><b>Name:</b> + TestgetVolume()</p> <p><b>Description:</b> Test getVolume() method in Instrument.class</p> <p><b>Input:</b> setUp("src/test/resources/system/demoDrumsComplex1.musicxml");</p> <pre> double actual = score.getParts().get(0).getInstruments().get("P1-I46").getVolume(); double expect = 78.7402; Assertions.assertEquals(expect, actual); </pre> <p><b>Output:</b> True</p>
<p><b>Name:</b> + TestgetPan()</p> <p><b>Description:</b> Test getPan() method in Instrument.class</p> <p><b>Input:</b> setUp("src/test/resources/system/demoDrumsComplex1.musicxml");</p> <pre> double actual = score.getParts().get(0).getInstruments().get("P1-I46").getPan(); double expect = 0; Assertions.assertEquals(expect, actual); </pre> <p><b>Output:</b> True</p>
<p><b>Name:</b> + TestAllSetter()</p> <p><b>Description:</b> Test AllSetter() methods in Instrument.class</p> <p><b>Input:</b> Instrument actual = new Instrument("P1-I45", "Pedal Hi-Hat");</p> <pre> actual.setMidiChannel(10); actual.setMidiProgram(1); actual.setMidiUnpitched(44); actual.setPan(0); actual.setVolume(8.7402); </pre>

```
int expectMidiChanel = 10;
int expectMidiProgram = 1;
int expectMidiUnpitched = 44;
double expectedPan = 0;
double expectedVolume = 8.7402;

    Assertions.assertEquals(expectMidiChanel,
actual.getMidiChannel());
    Assertions.assertEquals(expectMidiProgram,
actual.getMidiProgram());
    Assertions.assertEquals(expectMidiUnpitched,
actual.getMidiUnpitched());
    Assertions.assertEquals(expectedPan, actual.getPan());
    Assertions.assertEquals(expectedVolume, actual.getVolume());
```

**Output:** True



### 3.3 MeasureTest

**Measure**, on a music sheet, each part is divided into smaller bars and these are the measures. It is defined inside `<measure>` `</measure>`. A measure contains 7 attributes: number of divisions (defined inside `<divisions>` `</divisions>`), fifths ( defined in `<fifths>` `</fifths>`), time signature and time display (defined in `<time>` `</time>`), clef (defined in `<Clef>` `</Clef>`), number of staff lines (defined in `<staff-lines>` `</staff-lines>`, a tuning map (defined in `<tuning-step>` and `<tuning-octave>`), a list of notes.

#### Test cases:

**Name:** + testMeasure1()

**Description:** Test getDivisions() and getFifths() methods in Measure.class

**Input:** setUp("src/test/resources/system/demoDrumsSimple1.musicxml");  
 Measure m = score.getParts().get(0).getMeasures().get(0);  
 assertNotNull(m);

```
int d = m.getDivisions();
assertEquals(16,d);
```

```
int f = m.getFifths();
assertEquals(0,f);
```

**Output:** True

**Name:** + testMeasure2()

**Description:** Test getDivisions() method in Measure.class

**Input:** setUp("src/test/resources/system/demoDrumsSimple1.musicxml");  
 Measure m = score.getParts().get(0).getMeasures().get(1);  
 assertNotNull(m);

```
int d = m.getDivisions();
assertEquals(16,d);
```

**Output:** True

**Name:** +testMeasure3()

**Description:** Test getDivisions() and getFifths() methods in Measure.class

**Input:** setUp("src/test/resources/system/demoDrumsSimple2.musicxml");  
 Measure m = score.getParts().get(0).getMeasures().get(0);  
 assertNotNull(m);

```
int d = m.getDivisions(); //<divisions>32</divisions>
assertEquals(32,d);
```

```
int f = m.getFifths(); //<fifths>0</fifths>
assertEquals(0,f);
```

<b>Output:</b> True
<p><b>Name:</b> + testMeasure4()</p> <p><b>Description:</b> Test getDivisions() and getFifths() methods in Measure.class</p> <p><b>Input:</b> setUp("src/test/resources/system/demoDrumsSimple3.musicxml");</p> <pre>         Measure m = score.getParts().get(0).getMeasures().get(0); //measure number 1         assertNotNull(m);          int size = score.getParts().get(0).getMeasures().size(); //3         assertEquals(3,size);          int d = m.getDivisions(); //&lt;divisions&gt;16&lt;/divisions&gt;         assertEquals(16,d);          int f = m.getFifths(); //&lt;fifths&gt;0&lt;/fifths&gt;         assertEquals(0,f); </pre> <p><b>Output:</b> True</p>
<p><b>Name:</b> + TestgetIsRepeatStart()</p> <p><b>Description:</b> Test getIsRepeatStart() methods in Measure.class</p> <p><b>Input:</b> setUp("src/test/resources/system/money.musicxml");</p> <pre>         boolean actual = score.getParts().get(0).getMeasures().get(0).getIsRepeatStart();         assertTrue(actual); </pre> <p><b>Output:</b> True</p>
<p><b>Name:</b> + TestgetIsRepeatStop()</p> <p><b>Description:</b> Test getIsRepeatStop() methods in Measure.class</p> <p><b>Input:</b> setUp("src/test/resources/system/money.musicxml");</p> <pre>         boolean actual = score.getParts().get(0).getMeasures().get(0).getIsRepeatStop();         assertTrue(actual); </pre> <p><b>Output:</b> True</p>
<p><b>Name:</b> + TestgetBarLine()</p> <p><b>Description:</b> Test getBarLine() methods in Measure.class</p> <p><b>Input:</b> setUp("src/test/resources/system/money.musicxml");</p> <pre>         String actualStyle1 = score.getParts().get(0).getMeasures().get(0).getBarLineLeft().getStyle();         String expectStyle1 = "heavy-light";         assertEquals(actualStyle1,expectStyle1);          String actualRepeatType1 = score.getParts().get(0).getMeasures().get(0).getBarLineLeft().getRepeatType();         String expectRepeatType1 = "forward";         assertEquals(actualRepeatType1,expectRepeatType1); </pre>

```
assertNull(score.getParts().get(0).getMeasures().get(0).getBarLineLeft().getRepeatNum());
```

```
String actualStyle2 =
score.getParts().get(0).getMeasures().get(0).getBarLineRight().getStyle();
String expectStyle2 = "light-heavy";
assertEquals(actualStyle2,expectStyle2);
```

```
String actualRepeatType2 =
score.getParts().get(0).getMeasures().get(0).getBarLineRight().getRepeatType();
String expectRepeatType2 = "backward";
assertEquals(actualRepeatType2,expectRepeatType2);
```

```
int actualRepeatNumber2 =
score.getParts().get(0).getMeasures().get(0).getBarLineRight().getRepeatNum();
int expectRepeatNumber2 = 8;
assertEquals(actualRepeatNumber2,expectRepeatNumber2);
```

**Output:** True

**Name:** + TestgetDirections()

**Description:** Test getDirections() methods in Measure.class

**Input:** setUp("src/test/resources/system/money.musicxml");

```
String actual1 =
score.getParts().get(0).getMeasures().get(0).getDirections().get(0).getPlacement();
String expect1 = "above";
assertEquals(actual1,expect1);
```

```
String actual2 =
score.getParts().get(0).getMeasures().get(0).getDirections().get(0).getWords();
String expect2 = "x8";
assertEquals(actual2,expect2);
```

```
Double[] actual3 =
score.getParts().get(0).getMeasures().get(0).getDirections().get(0).getPosition();
Double[] expect3 = {0.0,0.0};
assertEquals(actual3[0],expect3[0]);
assertEquals(actual3[1],expect3[1]);
```

**Output:** True

**Name:** + TestgetTimeSignature()

**Description:** Test getTimeSignature() methods in Measure.class

**Input:** setUp("src/test/resources/system/money.musicxml");

```
int[] actual =
score.getParts().get(0).getMeasures().get(0).getTimeSignature();
int[] expect = {7,4};
assertEquals(actual[0],expect[0]);
assertEquals(actual[1],expect[1]);
```

```
assertTrue(score.getParts().get(0).getMeasures().get(0).getTimeDisplay());
Output: True
```

**Name:** + TestgetClef()

**Description:** Test getClef() methods in Measure.class

**Input:** setUp("src/test/resources/system/demoDrumsSimple3.musicxml");  
 Clef expect = new Clef('G',2);  
 Clef actual = score.getParts().get(0).getMeasures().get(0).getClef();  
 assertEquals(actual.getLine(),expect.getLine());  
 assertEquals(actual.getSymbol(),expect.getSymbol());

```
assertFalse(score.getParts().get(0).getMeasures().get(0).getTunedMeasure());
Output: True
```

**Name:** + TestgetStaffLines()

**Description:** Test getStaffLines() methods in Measure.class

**Input:** setUp("src/test/resources/system/money.musicxml");  
 int actual =  
 score.getParts().get(0).getMeasures().get(0).getStaffLines();  
 int expect = 4;  
 assertEquals(actual,expect);

**Output:** True

**Name:** + TestgetPercussion()

**Description:** Test getPercussion() methods in Measure.class

**Input:** setUp("src/test/resources/system/demoDrumsSimple3.musicxml");  
 assertTrue(score.getParts().get(0).getMeasures().get(0).getPercussion());  
 setUp("src/test/resources/system/demoGuitarComplex1.musicxml");

```
assertFalse(score.getParts().get(0).getMeasures().get(0).getPercussion());
Output: True
```

**Name:** + TestgetTab()

**Description:** Test TestgetTab() methods in Measure.class

**Input:** setUp("src/test/resources/system/demoGuitarComplex1.musicxml");  
 assertTrue(score.getParts().get(0).getMeasures().get(0).getTab());  
  
 setUp("src/test/resources/system/demoDrumsSimple1.musicxml");  
 assertFalse(score.getParts().get(0).getMeasures().get(0).getTab());

**Output:** True

### 3.4 NotationTest

**Notation**, a special attribute of certain notes in a song. It Is defined inside of `<notations>` `</notations>` and within `<note>`. This class contains 6 attributes: 3 Lists (each for Slur, Tied and Slide), position of the string and fret of the played note and ornament

#### Test cases:

**Name:** + notationTest1()

**Description:** This test use demoDrumsSimple1 as the test file. The goal is to check whether a notation exists within a note in each measure of the song

**Input:** setUp("src/test/resources/system/demoDrumsSimple1.musicxml");

```
boolean existNotation = false;
List<Note> noteList =
score.getParts().get(0).getMeasures().get(0).getNotes();
for(int i = 0; i < noteList.size(); i++) {
    if(noteList.get(i).getNotation() != null) {
        existNotation = true;
    }
}
Assertions.assertTrue(existNotation == true);
```

**Output:** True

**Name:** + notationTest2()

**Description:** This test use demoDrumsSimple1 as the test file. The goal is to check whether a notation exists within a note in each measure of the song

**Input:** setUp("src/test/resources/system/demoDrumsSimple1.musicxml");

```
boolean existNotation = false;
List<Note> noteList =
score.getParts().get(0).getMeasures().get(1).getNotes();
for(int i = 0; i < noteList.size(); i++) {
    if(noteList.get(i).getNotation() != null) {
        existNotation = true;
    }
}
Assertions.assertTrue(existNotation == true);
```

**Output:** True

**Name:** + notationTest3()

**Description:** This test use demoDrumsSimple1 as the test file. The goal is to check whether a notation exists within a note in each measure of the song

**Input:** setUp("src/test/resources/system/demoDrumsSimple1.musicxml");

```
boolean existNotation = false;
List<Note> noteList =
score.getParts().get(0).getMeasures().get(2).getNotes();
for(int i = 0; i < noteList.size(); i++) {
    if(noteList.get(i).getNotation() != null) {
```

<pre>                                 existNotation = true;                                 }                             }         Assertions.assertTrue(existNotation == false); <b>Output:</b> True </pre>
<p><b>Name:</b> + notationTest4()</p> <p><b>Description:</b> This test use demoDrumsSimple1 as the test file. The goal is to check whether a notation exists within a note in each measure of the song</p> <p><b>Input:</b> setUp("src/test/resources/system/demoDrumsSimple2.musicxml");</p> <pre>         Notation n1 = score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getNotation();         Notation n2 = score.getParts().get(0).getMeasures().get(0).getNotes().get(1).getNotation();         Notation n3 = score.getParts().get(0).getMeasures().get(0).getNotes().get(2).getNotation();         Notation n4 = score.getParts().get(0).getMeasures().get(0).getNotes().get(3).getNotation();         assertNull(n1);         assertNotNull(n2);         assertNotNull(n3);         assertNotNull(n4); </pre> <p><b>Output:</b> True</p>
<p><b>Name:</b> + notationTest5()</p> <p><b>Description:</b> This test use demoDrumsSimple1 as the test file. The goal is to check whether a notation exists within a note in each measure of the song</p> <p><b>Input:</b> setUp("src/test/resources/system/demoDrumsSimple1.musicxml");</p> <pre>         int numOfNotaion = 0;         List&lt;Note&gt; noteList1 = score.getParts().get(0).getMeasures().get(1).getNotes();         for(int i = 0; i &lt; noteList1.size(); i++) {             if(noteList1.get(i).getNotation() != null) {                 numOfNotaion++;             }         }         Assertions.assertTrue(numOfNotaion == 2);         numOfNotaion = 0;          List&lt;Note&gt; noteList2 = score.getParts().get(0).getMeasures().get(0).getNotes();         for(int i = 0; i &lt; noteList2.size(); i++) {             if(noteList2.get(i).getNotation() != null) {                 numOfNotaion++;             }         }         Assertions.assertTrue(numOfNotaion == 2); </pre> <p><b>Output:</b> True</p>

<p><b>Name:</b> + notationTestGetSlur()</p> <p><b>Description:</b> Test the getSlur() method in Notation.class</p> <p><b>Input:</b> setUp("src/test/resources/system/demoGuitarSimple4.musicxml");  List&lt;Slur&gt; list =  score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getNotation().getSlurs();  Assertions.assertTrue(list.size() == 1);</p> <p><b>Output:</b> True</p>
<p><b>Name:</b> +notationTestGetFret()</p> <p><b>Description:</b> Test the getFret() method in Notation.class</p> <p><b>Input:</b> setUp("src/test/resources/system/demoGuitarSimple4.musicxml");  int fret =  score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getNotation().getFret();  Assertions.assertTrue(fret == 3);</p> <p><b>Output:</b> True</p>
<p><b>Name:</b> + notationTestGetString()</p> <p><b>Description:</b> Test the getString() method in Notation.class</p> <p><b>Input:</b> setUp("src/test/resources/system/demoGuitarSimple4.musicxml");  int string =  score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getNotation().getString();  Assertions.assertTrue(string == 5);</p> <p><b>Output:</b> True</p>
<p><b>Name:</b> + notationTestGetTech()</p> <p><b>Description:</b> Test the getTech() method in Notation.class</p> <p><b>Input:</b> setUp("src/test/resources/system/demoGuitarComplex1.musicxml");  Technical tec1 =  score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getNotation().getTechnical();  Assertions.assertTrue(tec1.getHarmonic().equals("natural"));</p> <p>Technical tec2 =  score.getParts().get(0).getMeasures().get(1).getNotes().get(0).getNotation().getTechnical();  Assertions.assertEquals(tec2.getPullOff().get(0).getNumber(), 2);</p> <p>Assertions.assertTrue(tec2.getPullOff().get(0).getType().equals("start"));</p> <p>Assertions.assertTrue(tec2.getPullOff().get(0).getSymbol().equals("P"));</p> <p>Technical tec3 =  score.getParts().get(0).getMeasures().get(17).getNotes().get(11).getNotation().getTechnical();  Assertions.assertEquals(tec3.getHammerOns().get(0).getNumber(), 2);</p>

```
Assertions.assertTrue(tec3.getHammerOns().get(0).symbol().equals("H"));

Assertions.assertTrue(tec3.getHammerOns().get(0).type().equals("start"));

        setUp("src/test/resources/system/demoGuitarSimple1.musicxml");
        double actual =
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getNotation().getTechnical().getBend().getBendAlter();
        double expect = 2.0;
        Assertions.assertEquals(actual,expect);
```

**Output:** True

**Name:** + notationTestgetOrnaments()

**Description:** Test the getOrnaments() method in Notation.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");

```
        Ornament orn =
score.getParts().get(0).getMeasures().get(43).getNotes().get(14).getNotation().getOrnaments();
```

```
        Assertions.assertEquals(orn.getTremolo().getNumber(),1);
```

**Output:** True



### 3.5 NoteTest

**Note** Contains information about the note sound, position, duration, is it part of a chord, etc.

#### Test cases:

<p><b>Name:</b> + testNoteNumber()</p> <p><b>Description:</b> Test the total number of note by using its list</p> <p><b>Input:</b> setUp("src/test/resources/system/demoDrumsSimple1.musicxml");  List&lt;Note&gt; noteList1 =  score.getParts().get(0).getMeasures().get(0).getNotes();  Assertions.assertTrue(noteList1.size() == 13);</p> <p><b>Output:</b> True</p>
<p><b>Name:</b> + testGetNotaion()</p> <p><b>Description:</b> Test getNotation() method in Note.class</p> <p><b>Input:</b> setUp("src/test/resources/system/demoDrumsSimple2.musicxml");  List&lt;Note&gt; noteList1 =  score.getParts().get(0).getMeasures().get(0).getNotes();  Notation n1 =  score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getNotation();  Assertions.assertTrue(noteList1.get(0).getNotation() == n1);</p> <p><b>Output:</b> True</p>
<p><b>Name:</b> + testGetNoteType()</p> <p><b>Description:</b> Test getType() method in Note.class</p> <p><b>Input:</b> setUp("src/test/resources/system/demoDrumsSimple1.musicxml");  int notetype1 =  score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getType();  Assertions.assertEquals(notetype1, 8);</p> <p>int notetype2 =  score.getParts().get(0).getMeasures().get(0).getNotes().get(1).getType();  Assertions.assertEquals(notetype2, 8);</p> <p>setUp("src/test/resources/system/demoGuitarSimple4.musicxml");</p> <p>int notetype3 =  score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getType();  Assertions.assertEquals(notetype3, 16);</p> <p>int notetype4 =  score.getParts().get(0).getMeasures().get(0).getNotes().get(1).getType();  Assertions.assertEquals(notetype4, 8);</p> <p><b>Output:</b> True</p>
<p><b>Name:</b> + testGetNotedot()</p>

**Description:** Test getDot() method in Note.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");  
 int dotNum1 =  
 score.getParts().get(0).getMeasures().get(1).getNotes().get(0).getDot();  
 Assertions.assertEquals(dotNum1, 0);  
  
 int dotNum2 =  
 score.getParts().get(0).getMeasures().get(17).getNotes().get(0).getDot();  
 Assertions.assertEquals(dotNum2, 1);  
  
 int dotNum3 =  
 score.getParts().get(0).getMeasures().get(64).getNotes().get(2).getDot();  
 Assertions.assertEquals(dotNum3, 2);

**Output:** True

**Name:** + testGetNoteStem()

**Description:** Test getStem() method in Note.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");  
 String s1 =  
 score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getStem();  
 Assertions.assertTrue(s1.equals("up"));  
  
 setUp("src/test/resources/system/demoGuitarSimple5.musicxml");  
 String s2 =  
 score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getStem();  
 assertNull(s2);  
  
 setUp("src/test/resources/system/demoGuitarSimple4.musicxml");  
 String s3 =  
 score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getStem();  
 Assertions.assertTrue(s3.equals("none"));

**Output:** True

**Name:** + testGetNoteHead()

**Description:** Test getNotehead() method in Note.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");  
 String s1 =  
 score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getNotehead();  
 Assertions.assertTrue(s1.equals("x"));  
  
 setUp("src/test/resources/system/demoDrumsSimple1.musicxml");  
 String s2 =  
 score.getParts().get(0).getMeasures().get(0).getNotes().get(3).getNotehead();  
 Assertions.assertTrue(s2.equals("x"));  
  
 setUp("src/test/resources/system/demoGuitarSimple4.musicxml");  
 String s3 =  
 score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getNotehead();  
 assertNull(s3);

**Output:** True

**Name:** + TestgetParentheses()

**Description:** Test getParentheses() method in Note.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");

assertTrue(score.getParts().get(0).getMeasures().get(7).getNotes().get(11).getParentheses());

assertFalse(score.getParts().get(0).getMeasures().get(8).getNotes().get(0).getParentheses());

**Output:** True

**Name:** + TestetGrace()

**Description:** Test getGrace() method in Note.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");

assertTrue(score.getParts().get(0).getMeasures().get(7).getNotes().get(17).getGrace());

assertFalse(score.getParts().get(0).getMeasures().get(7).getNotes().get(16).getGrace());

**Output:** True

**Name:** + TestgetChord()

**Description:** Test getChord() method in Note.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");

assertTrue(score.getParts().get(0).getMeasures().get(8).getNotes().get(1).getChord());

assertFalse(score.getParts().get(0).getMeasures().get(8).getNotes().get(0).getChord());

**Output:** True

**Name:** + TestgetRest()

**Description:** Test getRest() method in Note.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");

assertTrue(score.getParts().get(0).getMeasures().get(64).getNotes().get(1).getRest());

assertFalse(score.getParts().get(0).getMeasures().get(65).getNotes().get(0).getRest());

**Output:** True

**Name:** + TestgetTimeModification()

**Description:** Test getTimeModification() method in Note.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");

HashMap<String, Integer> actual =

```
score.getParts().get(0).getMeasures().get(84).getNotes().get(0).getTimeModification();
```

```
int expect = 2;
assertEquals(actual.size(),expect);
assertEquals(actual.get("actual"),3);
assertEquals(actual.get("normal"),2);
```

**Output:** True

**Name:** + TestgetPitched()

**Description:** Test getPitched() method in Note.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");  
assertFalse(score.getParts().get(0).getMeasures().get(1).getNotes().get(0).getPitched());

**Output:** True

**Name:** + TestgetStep()

**Description:** Test getStep() method in Note.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");  
char actual =  
score.getParts().get(0).getMeasures().get(0).getNotes().get(8).getStep();  
char expected = 'G';  
assertEquals(actual,expected);

**Output:** True

**Name:** + TestgetOctave()

**Description:** Test getOctave() method in Note.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");  
int actual =  
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getOctave();  
int expected = 5;  
assertEquals(actual,expected);

**Output:** True

**Name:** +TestgetgetAlter()

**Description:** Test getAlter() method in Note.class

**Input:** setUp("src/test/resources/system/demoGuitarSimple1.musicxml");  
int actual =  
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getAlter();  
int expected = 1;  
assertEquals(actual,expected);

**Output:** True

**Name:** + TestgetDuration()

**Description:** Test getDuration() method in Note.class

**Input:** setUp("src/test/resources/system/demoGuitarSimple1.musicxml");  
int actual =  
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getDuration();  
int expected = 16;  
assertEquals(actual,expected);

**Output:** True

**Name:** + TestgetInstrumentID()

**Description:** Test getInstrumentID() method in Note.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");  
String actual =  
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getInstrumentID();  
String expected = "P1-I50";  
assertEquals(actual,expected);

**Output:** True

**Name:** + TesttoString()

**Description:** Test toString() method in Note.class

**Input:** setUp("src/test/resources/system/demoGuitarSimple1.musicxml");  
String expect = "{\n";  
expect += "step: " +  
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getStep() + "\n";  
expect += "octave: " +  
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getOctave() + "\n";  
expect += "duration: " +  
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getDuration() + "\n";  
  
expect += "type: " +  
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getType() + "\n";  
expect += "}";  
  
String actual =  
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).toString();  
assertEquals(actual,expect);

**Output:** True

### 3.6 PartTest

**Part** class contains 3 pieces of info of the melody: name of the part, a part id number and a list of different instruments. Each of the said info pieces also represent their respective attributes. The id is defined in `<score-part id = " "></score-part>`, the name is defined in `<part-name> </part-name>` and the instruments are represented with a map, and each instrument is mapped with an ID of their own

#### Test cases:

**Name:** + partTest1()

**Description:** To check the correctness of info related to a part such as the name of the part, its id, the total number of parts in the tested file, the number of measures and instruments in a designated part

**Input:** `setUp("src/test/resources/system/demoDrumsSimple1.musicxml");`  
`int partCount = 0;`  
`for(int i = 0; i < score.getParts().size(); i++) {`  
`partCount++;`  
`}`  
`Assertions.assertEquals(1, partCount);`  
`Assertions.assertTrue(score.getParts().get(0).getId().equals("P1"));`

`Assertions.assertTrue(score.getParts().get(0).getName().equals("Drumset"));`

`int measureCount = 0;`  
`for(int i = 0; i < score.getParts().get(0).getMeasures().size(); i++) {`  
`measureCount++;`  
`}`  
`Assertions.assertEquals(3, measureCount);`

`int instrumentCount = 0;`  
`for(int i = 0; i < score.getParts().get(0).getInstruments().size(); i++) {`  
`instrumentCount++;`  
`}`  
`Assertions.assertEquals(13, instrumentCount);`

**Output:** True

**Name:** + partTest2()

**Description:** To check the correctness of info related to a part such as the name of the part, its id, the total number of parts in the tested file, the number of measures and instruments in a designated part

**Input:** `setUp("src/test/resources/system/demoBassSimple1.musicxml");`  
`int partCount = 0;`  
`for(int i = 0; i < score.getParts().size(); i++) {`  
`partCount++;`

```

    }
    Assertions.assertEquals(1, partCount);
    Assertions.assertTrue(score.getParts().get(0).getId().equals("P1"));

    Assertions.assertFalse(score.getParts().get(0).getName().equals("Drumset"));

    Assertions.assertFalse(score.getParts().get(0).getName().equals("Lololol"));

    Assertions.assertTrue(score.getParts().get(0).getName().equals("Bass"));

    int measureCount = 0;
    for(int i = 0; i < score.getParts().get(0).getMeasures().size(); i++) {
        measureCount++;
    }
    Assertions.assertEquals(4, measureCount);

    int instrumentCount = 0;
    for(int i = 0; i < score.getParts().get(0).getInstruments().size(); i++) {
        instrumentCount++;
    }
    Assertions.assertEquals(0, instrumentCount);

```

**Output:** True

**Name:** + partTest3()

**Description:** To check the correctness of info related to a part such as the name of the part, its id, the total number of parts in the tested file, the number of measures and instruments in a designated part

**Input:** setUp("src/test/resources/system/demoDrumsSimple2.musicxml");  
 assertNotNull(score.getParts());  
 assertEquals("P1",score.getParts().get(0).getId());  
 assertEquals("Drumset",score.getParts().get(0).getName());

**Output:** True

**Name:** + partTest4()

**Description:** To check the correctness of info related to a part such as the name of the part, its id, the total number of parts in the tested file, the number of measures and instruments in a designated part

**Input:** setUp("src/test/resources/system/demoDrumsSimple3.musicxml");  
 assertNotNull(score.getParts());  
 assertEquals("Drumset",score.getParts().get(0).getName());  
 assertEquals("P1",score.getParts().get(0).getId());

**Output:** True

**Name:** + TestgetInstruments()

**Description:** Test getInstruments() method in Part.class

**Input:** setUp("src/test/resources/system/demoDrumsComplex1.musicxml");  
 int actualNumOfInstrument  
 =score.getParts().get(0).getInstruments().size();

<pre>int expectNumberOfInstrument = 13; assertEquals(actualNumOfInstrument,expectNumberOfInstrument);</pre> <b>Output:</b> True
<p><b>Name:</b> + TestgetMeasures()</p> <p><b>Description:</b> Test getMeasures() method in Part.class</p> <p><b>Input:</b> setUp("src/test/resources/system/demoDrumsComplex1.musicxml");</p> <pre>int actualNumOfMeasures =score.getParts().get(0).getMeasures().size(); int expectNumberOfMeasures = 126; assertEquals(actualNumOfMeasures,expectNumberOfMeasures);</pre> <b>Output:</b> True



### 3.7 ScoreTest

**Score Class:** Contains all the data including title and author, and partList. All classes used for XML parsing are, in some form or another, accessible via an object of the type Score.

**Test Cases:**

**Name:** scoreTest1()

**Description:** Tests the correct parsing of the score title, and the number of parts.

**Input Used:** src/test/resources/system.demoTitleAuthorTest.musicxml

**Derivation:** Matches the title of the music in the musicxml file under the element <movement-title> with the title parsed by the Score.

**Implementation:** Create and save a musicxml file which contains the <movement-title> element with the content "Title21". Using the getTitle() method of the Score, assert whether the returned value is "Title21". Since guitar and string instrument's musicxml files only contain one <part>, we always check if the number of Part objects generated by the Score is equal to 1.

**Result:** PASS

**Name:** scoreTest2()

**Description:** Tests the correct parsing of the score's artist.

**Input Used:** src/test/resources/system.demoTitleAuthorTest1.musicxml

**Derivation:** Matches the artist of the music in the musicxml file under the element <creator> with the artist parsed by the Score.

**Implementation:** Create and save a musicxml file which contains the <creator> element with the content "The Unforgiven". Using the getAuthor() method of the Score, assert whether the returned value is "The Unforgiven".

**Result:** PASS

**Name:** scoreTest3()

**Description:** Tests to ensure no Exception occurs if information is absent from the musicxml file.

**Input Used:** src/test/resources/system.demoGuitarSimple1.musicxml

**Derivation:** Attempt to retrieve the author and title values, when they are not present in the data.

**Implementation:** Create and save a musicxml file which contains neither the `<creator>` element nor the `<movement-title>` element. Use the `getAuthor()` and `getTitle()` methods to ensure no Exception has occurred and to check that the returned values are null.

**Result:** PASS

### **Sufficiency of Tests:**

All methods of Score are tested (`getTitle()`, `getAuthor()`, and `getParts()`). Since their corresponding .musicxml elements appear in a regular fashion (either at the top of the file, or not present), only 3 test cases are sufficient.

### 3.8 SlideTest

#### Slide class:

- One of the 3 special notations (slur, slide, tied) that could be assigned to a note and its adjacent partner. It is defined within the tags `<slide>` `</slide>`, which itself resides within the `<notations>` `</notations>`.
- A Slide object has 2 attributes:
  - **type** ( could be “start” or “stop” )
  - **number** ( the index of that current notation).

*Note:* The **number** attribute is not useful in this implementation of the program, thus it is not tested.

#### Tests Cases:

**Name:** slideTest5()

**Description:** Tests the distinction of the Notation element and the Slide element as well as the distinction of the Slide element from similar elements such as the Tied element during the XML parsing

**Input Used:** src/test/resources/system.demoGuitarSimple1.musicxml

**Derivation:** The input file has most regular notes. But the 12th and 13th notes have a Tied element. Ensure that all the first 11 notes don't have any Notation object, and that the last 2 notes have an associated Notation object which contains initialized lists of Tied, Slur and Slide. Make sure that the list of Tied objects for the last 2 notes are non-empty, but the Slide list is empty.

**Implementation:** Select a musicxml file with its 1st `<measure>` element containing 13 `<note>` elements. First ensure that all notes have been successfully parsed by checking if the corresponding Measure object has a List of Notes the size of 13, using the Measure's `getNotes()` method.

Then check if the first 11 Note objects throw a `NullPointerException` when trying to access its Slide object because the Note must first call its `getNotation()` which would return null because the first 11 notes don't have a corresponding `<notation>` element in the file. Then attempting to access the list of Slide objects using the `getSlides()` method is expected to throw the `NullPointerException` since the Notation object is null.

For the last 2 Note objects in the list, check if they have non-null Tied elements using the `getNotation().getTieds().get(0)`. It is expected that non-null Tied objects are returned by both the 12th and 13th `<note>` because they both have `<notation>` elements containing single `<tied>` elements.

Lastly, check for the last 2 Note objects in the list that their list of Slide objects is initialized, but are empty using the *getNotation().getSlides.get(0)*. Since the last 2 <note> elements in the file contain the <notation> element, it is expected that their List of Slides are initialized. But the <notation> element contains no <slide> elements, so the List should be empty. It is expected that on attempting to call the *get(0)* method on the List of Slides, that an *IndexOutOfBoundsException* occurs.

**Result:** PASS

**Name:** slideTest6()

**Description:** Tests the distinction of the Notation element and the Slide element as well as the distinction of the Slide element from similar elements such as the Slur element during the XML parsing. [Identical with previous test case with the exception that the Class being compared to is Slur instead of Tied]

**Input Used:** src/test/resources/system.demoGuitarComplex1.musicxml

**Derivation:** The input is a musicxml file which has a 2nd <measure> containing 12 <note> elements. The first 2 <note> elements contain a <notation> element with a <slur> element, but the remainder of the <measure> element contains no other <tied>, <slide> or <slur> elements.

**Implementation:** Identical to slurTest5(), except reference to the Tied class is replaced with that of the Slur class. As such the *getSlurs()* method is used in the stead of *getTieds()*.

**Result:** PASS

**Name:** slurTest7()

**Description:** Tests that a corresponding Slide object is always created corresponding to the <slide> element in the .musicxml file. This Slide object is always contained in the Notation attribute of the correct note, and whether its *type* attribute is "stop" or "start" is also recorded correctly.

**Input Used:** src/test/resources/system.demoGuitarComplex1.musicxml

**Derivation:** For each <slide> element in the file match it with a corresponding Slide object with attribute *type*="stop" within the correct Note object. Similarly for all Slide objects that should have the attribute *type*="start". Whether the Slide object is contained within the correct note, is determined by 0-indexed counting of the <measure> elements, and within each measure 0-indexed counting of the <note> elements.

**Implementation:** Create 2 Lists of Integer pairs. The first list called “slideStarts”, will contain the pair values of (measure index, note index within measure) for each <slide> element with the attribute *type*=”start”. The second list called “slideEnds” will contain the same data, but for all <slide> elements with the attribute *type*=”stop” instead.

Iterate through each Measure object in the score using the List *score.getParts().get(0).getMeasures()*. Then iterate through each Note object within the measure using the list *measures.get(i).getNotes()*. Thus, for the *ith* Measure and the *jth* Note object check if it contains a Slide object with attribute *type*=”start”. If so, then check if the first element of the “slideStarts” List has the pair (*i, j*). If the attribute was *type*=”stop”, then check the “slideEnds” List for the pair (*i, j*). It is expected that the List always contains these pairs. Keep a counter for both arrays to access the next element and keep a count of the number of matches so far.

At the end it is checked if the number of starting Slide objects is the same as the stopping Slide objects, and that the number of matches corresponds to the size of the both lists. Both these values are expected to be true.

*[Exhaustive Testing]*

**Result:** PASS

### **Sufficiency of Tests:**

The program mainly requires to identify the note that this Slide object corresponds to and whether it is the start or the end of the Slide element in order to display this notation correctly. Thus, it is sufficient to exhaustively test for randomly occurring slides in a relatively long .musicxml file to see if the corresponding notes have the required Slide of the correct type.

### 3.9 SlurTest

#### Slur Class:

- One of the 3 special notations (slur, slide, tide) that could be assigned to a note and its adjacent one. It is defined in `<slur>` `</slur>` and within `<notations>` `</notations>`.
- A Slur object has 3 attributes:
  - **type** ( could be “start” or “stop” )
  - **placement** (could be “above” or “below”)
  - **number** (the index of that current notation)

*Note:* Only the **type** attribute was used in the implementation, so only it is tested.

#### Test Cases:

**Name:** slurTest1()

**Description:** Tests the distinction of the Notation element and the Slur element as well as the distinction of the Slur element from similar elements such as the Tied element during the XML parsing

**Input Used:** src/test/resources/system.demoGuitarSimple1.musicxml

**Derivation:** The input file has most regular notes. But the 12th and 13th notes have a Tied element. Ensure that all the first 11 notes don't have any Notation object, and that the last 2 notes have an associated Notation object which contains initialized lists of Tied, Slur and Slide. Make sure that the list of Tied objects for the last 2 notes are non-empty, but the Slur list is empty.

**Implementation:** Identical file, `<measure>` and `<note>` elements as **slideTest5()**, except the `NullPointerException` (for the first 11 notes of the measure) and the `IndexOutOfBoundsException` (for the 12th and the 13th notes) are tested for the Slur object instead.

**Result:** PASS

**Name:** slurTest2()

**Description:** Tests that a corresponding Slur object is always created corresponding to the `<slur>` element in the .musicxml file. This Slur object is always contained in the Notation attribute of the correct note, and whether its *type* attribute is “stop” or “start” is also recorded correctly.

**Input Used:** src/test/resources/system.demoDrumsComplex1.musicxml

**Derivation:** For each <slur> element in the file match it with a corresponding Slur object with attribute *type*="stop" within the correct Note object. Similarly for all Slur objects that should have the attribute *type*="start". Whether the Slur object is contained within the correct note, is determined by 0-indexed counting of the <measure> elements, and within each measure 0-indexed counting of the <note> elements.

**Implementation:** Identical to **slideTest7()**, except the 2 List of Integer pairs is generated for every <slur> element in the .musicxml file, and they are matched with all the Slur objects generated by the XML parsing.

**Result:** PASS

### Sufficiency of Tests:

The program mainly requires to identify the note that this Slur object corresponds to and whether it is the start or the end of the Slur element in order to display this notation correctly. Thus, it is sufficient to exhaustively test for randomly occurring slurs in a relatively long .musicxml file to see if the corresponding notes have the required Slur of the correct type.

### 3.10 TiedTest

#### Tied Class:

- One of the 3 special notations (slur, slide, tide) that could be assigned to a note and its adjacent one. It is defined in `<Tied> </Tied>` and within `<notations> </notations>`.
- A Tied object has 3 attributes:
  - **type** ( could be “start” or “stop” ),
  - **placement** (could be “above” or “below”)
  - **number** (the index of that current notation)

*Note:* Only the **type** attribute is used, therefore only its value is tested.

#### Tests Cases:

**Name:** tiedTest1()

**Description:** Tests that a corresponding Tied object is always created corresponding to the `<tied>` element in the .musicxml file. This Tied object is always contained in the Notation attribute of the correct note, and whether its *type* attribute is “stop” or “start” is also recorded correctly.

**Input Used:** src/test/resources/system.demoDrumsComplex1.musicxml

**Derivation:** For each `<tied>` element in the file match it with a corresponding Tied object with attribute *type*=“stop” within the correct Note object. Similarly for all Tied objects that should have the attribute *type*=“start”. Whether the Tied object is contained within the correct note, is determined by 0-indexed counting of the `<measure>` elements, and within each measure 0-indexed counting of the `<note>` elements.

**Implementation:** Identical to **slideTest7()**, except the 2 List of Integer pairs is generated for every `<tied>` element in the .musicxml file, and they are matched with all the Tied objects generated by the XML parsing.

**Result:** PASS

#### Sufficiency of Tests:

The program mainly requires to identify the note that this Tied object corresponds to and whether it is the start or the end of the Tied element in order to display this notation correctly. Thus, it is sufficient to exhaustively test for randomly occurring ties in a relatively long .musicxml file to see if the corresponding notes have the required Tied of the correct type. The testing for the relationships of the Notation, Tied, Slur, and Slide are already completed by other tests.



## 4. Model Tests

The following tests ensure that the music string pattern generation and the sheet music generation are done correctly

### 4.1 Beaming Test

Uses a similar setUp() method as the previous tests. This setUp() method also initializes a *SheetScore* object to generate the sheet music GUI elements. The purpose of these tests is to ensure that the output beam structures are represented accurately.

#### Test Cases:

**Name:** testCorrectBeamingGroups1() - testCorrectBeamingGroups3()

**Description:** Tests for the correct grouping of beams (i.e. which notes are beamed together in a group).

**Input Used:** src/test/resources/system/demoGuitarComplex1.musicxml  
src/test/resources/system.demoDrumsComplex1.musicxml

**Derivation:** Cross-referencing MuseScore, check which notes need to be beamed together in a measure

**Implementation:** Selecting a single *MusicMeasure* (GUI element for <measure>), generate a *MeasureBeamData* object. This object stores the list of integers representing the beams as follows: 0 = not a beamed note, 1 = note belongs to the first collection of beamed notes, 2 = note belongs to the 2nd collection of beamed notes, so on...). Then create a list of Integers manually which correctly represents these integers by referencing MuseScore as the ideal result. Now compare the values in the list created manually and the *MeasureBeamData* List of Integers which we receive using the *getBeamNumbers()* method. The expected result is that all the List elements match.

**Result:** PASS

**Name:** testCorrectBeamDrawing1() - testCorrectBeamDrawing2()

**Description:** Tests for the correct number of connections between beamed notes (the horizontal lines), and ensure that the number of full length and half length connections are accurate.

**Input Used:** src/test/resources/system/demoGuitarComplex1.musicxml  
src/test/resources/system.demoDrumsComplex1.musicxml

**Derivation:** Cross-referencing MuseScore, check how many connections are required for each collection of beamed notes.

**Implementation:** Selecting a single *MusicMeasure* (GUI element for <measure>), generate a *MeasureBeamData* object and use its data to create a *BeamInfoProcessor* object. For each beamed note in a measure this object contains a pair of numbers which it stores in a List for all the notes. This List can be obtained with the method *getBeamData()*. The pairs (a, b) represent “a = the number of full length connections of a beam to the next note, b = the number of half length connections of a beam to the next note.” The only exception is when the note is at the end of a beamed group of notes, where these numbers indicate the number of beams from this end note to the previous note. Now we manually create a List of these pairs by referencing the MuseScore output for the number of full-length and half-length connections for each beamed note. These values are compared with that of the list in the *BeamInfoProcessor* object. It is expected that these lists are identical.

**Result:** PASS

### Sufficiency of Tests:

The system requires 3 things to ensure that the beams are correctly generated: which notes are connected to each other with beams, which notes are beamed separately from the subsequent notes, and how many connecting lines are required for a beam. The first 2 requirements are checked by the 1st test, and the last requirement is checked by the last test case.

## 4. 2 PatternString Test

Uses a similar setUp() method as the previous tests. This setUp() method also initializes a *SheetScore* object to generate the sheet music GUI elements. Then it uses the *musicPlayer* constructor to generate a JFugue Pattern string, which represents the music playback in a form comprehensible by the JFugue library. The purpose of these tests is to ensure that this string is correctly generated so music is played correctly.

### Test Cases:

**Name:** checkChording

**Description:** Tests for correct implementation of the chords within the JFugue String.

**Input Used:** src/test/resources/system/demoGuitarComplex1.musicxml

**Derivation:** Cross-referencing MuseScore, check which notes need to be chorded together. Then check if the pattern string has the correct elements to represent these chords.

**Implementation:** The JFugue string requires that notes which are in a chord with one another have the “+” symbol between them, and that measure terminations are signified with “|”. So an array is manually created which stores the number of such “+” signs required where each element represents the number of “+” signs in a measure. (For each chord there are  $n - 1$  “+” symbols if the chord has  $n$  notes). Then we scan through the string using a scanner and incrementing another array each time a “+” is seen. This list stores for the *ith* element the number of “+” signs in the *ith* measure of the pattern string. Then, these lists are compared. It is expected that they are the same.

**Result:** PASS

**Name:** checkDotage()

**Description:** Tests for correct implementation of the dots within the JFugue String.

**Input Used:** src/test/resources/system/demoGuitarComplex1.musicxml

**Derivation:** Cross-referencing MuseScore, check which notes need to have dots. Then check if the pattern string has the correct elements to represent these dots.

**Implementation:** The JFugue string requires that notes which have dots with the

“.” (period character), and that measure terminations are signified with “|”. So an array is manually created which stores the number of such “.” signs required where each element represents the number of “.” signs in a measure. (For each dot for a note, there is one “.” character). Then we scan through the string using a scanner and incrementing another array each time a “.” is seen. This list stores for the *ith* element the number of “.” signs in the *ith* measure of the pattern string. Then, these lists are compared. It is expected that they are the same meaning the measures have the same number of dots.

**Result:** PASS

## 5. Test Coverage Analysis

### 4.1 Test Summary

#### Test Summary

106 tests	0 failures	5 ignored	2m56.35s duration	100% successful
--------------	---------------	--------------	----------------------	--------------------

Ignored tests Packages **Classes**

Class	Tests	Failures	Ignored	Duration	Success rate
<a href="#">GUI.GuiTest</a>	14	0	0	2m16.58s	100%
<a href="#">Test_Coverage.ClefTest</a>	5	0	0	3.848s	100%
<a href="#">Test_Coverage.InstrumentTest</a>	11	0	0	8.872s	100%
<a href="#">Test_Coverage.MeasureTest</a>	13	0	0	1.654s	100%
<a href="#">Test_Coverage.NotationTest</a>	10	0	0	1.988s	100%
<a href="#">Test_Coverage.NoteTest</a>	18	0	0	13.940s	100%
<a href="#">Test_Coverage.OrnamentTest</a>	1	0	0	1.164s	100%
<a href="#">Test_Coverage.PartTest</a>	6	0	0	2.271s	100%
<a href="#">Test_Coverage.ScoreTest</a>	2	0	0	0.051s	100%
<a href="#">Test_Coverage.SlideTest</a>	8	0	0	2.131s	100%
<a href="#">Test_Coverage.SlurTest</a>	3	0	0	2.022s	100%
<a href="#">Test_Coverage.StaffTest</a>	2	0	0	0.055s	100%
<a href="#">Test_Coverage.TiedTest</a>	3	0	0	1.306s	100%
<a href="#">converter.ScoreTest</a>	4	0	1	0.454s	100%
<a href="#">converter.TabRowTest</a>	1	0	1	0s	-
<a href="#">converter.measure_line.InsidesPatternTest</a>	2	0	2	0s	-
<a href="#">converter.notes.NoteFactoryTest</a>	2	0	0	0.017s	100%
<a href="#">system.SampleInputs</a>	1	0	1	0s	-

Generated by [Gradle 7.3.3](#) at Apr 11, 2022, 10:48:30 PM

**Summary:** In total, 106 test cases are successfully tested in 3 minutes almost. Including JUnit test and JavaFX GUI test.

### 4.2 Test Coverage

#### For whole Packages

TAB2XML\_G14

#### TAB2XML\_G14

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
<a href="#">custom_model</a>	<div><div></div></div>	40%	<div><div></div></div>	34%	305	434	892	1,552	65	128	12	25
<a href="#">custom_model.note</a>	<div><div></div></div>	7%	<div><div></div></div>	2%	187	208	494	551	63	82	5	8
<a href="#">GUI</a>	<div><div></div></div>	38%	<div><div></div></div>	22%	154	222	507	824	68	121	8	16
<a href="#">converter.note</a>	<div><div></div></div>	43%	<div><div></div></div>	36%	196	299	342	687	63	125	4	14
<a href="#">custom_player</a>	<div><div></div></div>	49%	<div><div></div></div>	34%	86	130	155	343	5	28	0	1
<a href="#">converter.instruction</a>	<div><div></div></div>	38%	<div><div></div></div>	29%	76	101	133	222	14	35	1	6
<a href="#">converter</a>	<div><div></div></div>	72%	<div><div></div></div>	67%	61	153	99	431	6	49	0	7
<a href="#">custom_model.notehead</a>	<div><div></div></div>	0%	<div><div></div></div>	0%	16	16	122	122	9	9	6	6
<a href="#">custom_model.rest</a>	<div><div></div></div>	0%	<div><div></div></div>	0%	13	13	92	92	8	8	6	6
<a href="#">converter.measure</a>	<div><div></div></div>	83%	<div><div></div></div>	68%	57	145	64	390	6	42	0	4
<a href="#">models.measure.note</a>	<div><div></div></div>	17%	<div><div></div></div>	50%	59	71	103	123	58	69	9	11
<a href="#">models.measure.note.notations.technical</a>	<div><div></div></div>	11%	<div><div></div></div>	0%	48	57	95	106	44	53	8	9
<a href="#">custom_component_data</a>	<div><div></div></div>	91%	<div><div></div></div>	72%	52	209	43	469	8	120	0	19
<a href="#">models.measure.note.notations</a>	<div><div></div></div>	13%	<div><div></div></div>	0%	29	36	59	67	25	32	3	4
<a href="#">utility</a>	<div><div></div></div>	92%	<div><div></div></div>	79%	35	115	45	336	19	67	1	12
<a href="#">converter.measure_line</a>	<div><div></div></div>	69%	<div><div></div></div>	50%	15	31	23	73	2	13	0	4
<a href="#">models.part_list</a>	<div><div></div></div>	24%	<div><div></div></div>	n/a	17	24	39	51	17	24	3	5
<a href="#">models</a>	<div><div></div></div>	61%	<div><div></div></div>	n/a	14	33	27	62	14	33	1	5
<a href="#">models.measure.attributes</a>	<div><div></div></div>	73%	<div><div></div></div>	n/a	10	36	20	66	10	36	0	6
<a href="#">models.measure.barline</a>	<div><div></div></div>	65%	<div><div></div></div>	n/a	7	21	11	30	7	21	1	3
<a href="#">models.measure</a>	<div><div></div></div>	67%	<div><div></div></div>	n/a	5	18	10	28	5	18	1	2
<a href="#">custom_exceptions</a>	<div><div></div></div>	0%	<div><div></div></div>	n/a	4	4	8	8	4	4	4	4
<a href="#">models.measure.direction</a>	<div><div></div></div>	84%	<div><div></div></div>	n/a	2	15	4	21	2	15	0	3
Total	16,420 of 31,686	48%	1,459 of 2,434	40%	1,448	2,391	3,387	6,654	522	1,132	73	180

## For customer\_component\_data package

TAB2XML\_G14 &gt; custom\_component\_data

### custom\_component\_data

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Measure		82%		68%	12	43	19	118	1	19	0	1
Note		93%		88%	8	40	4	87	4	22	0	1
Technical		90%		81%	2	13	3	25	0	5	0	1
Score		90%		83%	1	7	6	30	0	4	0	1
Part		96%		72%	5	14	4	49	0	5	0	1
Tied		81%		50%	2	6	1	10	0	4	0	1
Bend		75%		50%	2	4	1	6	1	3	0	1
BarLine		92%		62%	4	9	1	12	1	5	0	1
Clef		86%		50%	4	8	3	14	0	4	0	1
Direction		94%		50%	4	8	0	10	0	4	0	1
Tremolo		85%		50%	2	4	1	6	1	3	0	1
HammerOn		95%		50%	2	6	0	8	0	4	0	1
PullOff		95%		50%	2	6	0	8	0	4	0	1
Slide		96%		50%	1	4	0	6	0	3	0	1
Notation		100%		100%	0	13	0	30	0	8	0	1
Instrument		100%		n/a	0	14	0	31	0	14	0	1
Slur		100%		n/a	0	4	0	8	0	4	0	1
Ornament		100%		50%	1	3	0	5	0	2	0	1
StaffTuning		100%		n/a	0	3	0	6	0	3	0	1
Total	203 of 2,272	91%	49 of 178	72%	52	209	43	469	8	120	0	19

## For GUI package

TAB2XML\_G14 &gt; GUI

### GUI

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
MainViewController		28%		2%	35	46	150	215	13	24	0	1
CurrentSongSettingsWindowController		0%		0%	25	25	73	73	11	11	1	1
SystemDefaultSettingsWindowController		0%		0%	14	14	53	53	9	9	1	1
Highlighter		57%		35%	19	30	39	86	4	10	0	1
SaveMXLController		13%		0%	15	19	35	43	3	7	0	1
LoadScreen		0%		0%	6	6	33	33	4	4	1	1
ShowMXLController		25%		0%	7	11	25	35	4	8	0	1
PreviewController		83%		87%	5	22	26	126	4	18	0	1
SplashFXMLController.SplashScreen.new Runnable() { }		0%		n/a	3	3	18	18	3	3	1	1
SplashFXMLController.SplashScreen		0%		0%	5	5	16	16	3	3	1	1
MainApp		0%		n/a	4	4	15	15	4	4	1	1
NameChoosingController		0%		0%	6	6	11	11	4	4	1	1
DisplaySettingController		91%		69%	8	23	8	80	0	10	0	1
SplashFXMLController		0%		n/a	2	2	4	4	2	2	1	1
PreviewController.new Thread() { }		84%		100%	0	3	2	9	0	2	0	1
MainViewController.new Task() { }		100%		100%	0	3	0	10	0	2	0	1
Total	2,161 of 3,508	38%	150 of 193	22%	154	222	507	824	68	121	8	16

## For custom\_model package

TAB2XML\_G14 &gt; custom\_model

### custom\_model

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
StaffMeasure		0%		0%	61	61	170	170	2	2	1	1
TabMeasure		48%		32%	61	75	144	257	0	2	0	1
SheetScore		56%		58%	27	57	73	209	4	20	0	1
BeamInfoProcessor		41%		42%	19	31	62	111	1	5	0	1
ChordedSlur		0%		0%	21	21	84	84	9	9	1	1
MeasureBeamData		49%		53%	26	41	52	108	4	8	0	1
ArcLine		0%		0%	12	12	50	50	7	7	1	1
TimeModificationLabel		0%		0%	12	12	39	39	3	3	1	1
Bend		0%		0%	10	10	49	49	8	8	1	1
Tremolo		0%		0%	9	9	32	32	6	6	1	1
MusicMeasure		82%		63%	16	40	23	162	5	21	0	1
ExampleClass		0%		n/a	1	1	38	38	1	1	1	1
TabNoteStem		35%		35%	7	9	20	35	0	2	0	1
SlideLink		0%		n/a	7	7	21	21	7	7	1	1
PlaybackGUILinker		82%		70%	7	17	12	63	0	2	0	1
MainGraphicsTest		0%		n/a	2	2	9	9	2	2	1	1
SheetScore.TiedPair		0%		n/a	1	1	4	4	1	1	1	1
MainTester		0%		n/a	2	2	3	3	2	2	1	1
MeasureBeamData.BeamInfo		94%		100%	1	9	2	22	1	8	0	1
ScoreLine		97%		100%	1	6	1	22	1	3	0	1
SlantLine		95%		50%	1	2	1	11	0	1	0	1
MusicMeasure.new Thread() { }		93%		100%	0	3	2	11	0	2	0	1
NoteLinker		0%		n/a	1	1	1	1	1	1	1	1
RepeatBarLine		100%		n/a	0	2	0	33	0	2	0	1
BeamInfoProcessor.BeamCreationData		100%		n/a	0	3	0	9	0	3	0	1
Total	4,998 of 8,367	40%	403 of 612	34%	305	434	892	1,552	65	128	12	25

**Summary:** The test coverage for each package is high enough. And for detailed graph, please go “build/customJacocoReportDir/test/html/index.html”