Test Document

EECS 2311 Group 1

Abdelrahman Altamimi

Hieu Le

Mahdiar Shoraka

Prabjot Dhaliwal

Yongjie Ba

Table of contents

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)
Descripion: 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);


        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("   7/4\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---|-------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---|-------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\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: +cleftConstructorTest1()
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 geSymbol() 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 quals() 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-intrument 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"
+ "\tvol: " + 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");

int actual =
score.getParts().get(0).getInstruments().get("P1-I46").getMidiChannel();
int expect = 10;
Assertions.assertEquals(expect, actual);
Output: True

Name: + TestgetMidiProgram()
Description: Test getMidiProgram() method in Instrument.class
Input: setUp("src/test/resources/system/demoDrumsComplex1.musicxml");
int actual =
score.getParts().get(0).getInstruments().get("P1-I46").getMidiProgram();
int expect = 1;
Assertions.assertEquals(expect, actual);
Output: True

Name: + TestgetMidiUnpitched()
Description: Test getMidiUnpitched() method in Instrument.class
Input: setUp("src/test/resources/system/demoDrumsComplex1.musicxml");
int actual =
score.getParts().get(0).getInstruments().get("P1-I46").getMidiUnpitched();
int expect = 46;
Assertions.assertEquals(expect, actual);
Output: True

Name: + TestgetVolume()
Description: Test getVolume() method in Instrument.class
Input: setUp("src/test/resources/system/demoDrumsComplex1.musicxml");
double actual =
score.getParts().get(0).getInstruments().get("P1-I46").getVolume();
double expect = 78.7402;
Assertions.assertEquals(expect, actual);
Output: True

Name: + TestgetPan()
Description: Test getPan() method in Instrument.class
Input: setUp("src/test/resources/system/demoDrumsComplex1.musicxml");
double actual =
score.getParts().get(0).getInstruments().get("P1-I46").getPan();
double expect = 0;
Assertions.assertEquals(expect, actual);
Output: True

Name: + TestAllSetter()
Description: Test AllSetter() methods in Instrument.class
Input: Instrument actual = new Instrument("P1-I45", "Pedal Hi-Hat");
actual.setMidiChannel(10);
actual.setMidiProgram(1);
actual.setMidiUnpitched(44);
actual.setPan(0);
actual.setVolume(8.7402);

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

                Assertions.assertEquals(expectMidiChanel,
actual.getMidiChannel());
                Assertions.assertEquals(expectMidiProgram,
actual.getMidiProgram());
                Assertions.assertEquals(expectMidiUnipitched,
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);
```

Output: True

Name: + testMeasure4()
Description: Test getDivisions() and getFifths() methods in Measure.class
Input: setUp("src/test/resources/system/demoDrumsSimple3.musicxml");
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(); //<divisions>16</divisions>
assertEquals(16,d);

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

Name: + TestgetIsRepeatStart()
Description: Test getIsRepeatStart() methods in Measure.class
Input: setUp("src/test/resources/system/money.musicxml");
boolean actual =
score.getParts().get(0).getMeasures().get(0).getIsRepeatStart();
assertTrue(actual);
Output: True

Name: + TestgetIsRepeatStop()
Description: Test getIsRepeatStop() methods in Measure.class

Input: setUp("src/test/resources/system/money.musicxml");
boolean actual =
score.getParts().get(0).getMeasures().get(0).getIsRepeatStop();
assertTrue(actual);

Output: True

Name: + TestgetBarLine()
Description: Test getBarLine() methods in Measure.class
Input: setUp("src/test/resources/system/money.musicxml");
String acturalStyle1 =
score.getParts().get(0).getMeasures().get(0).getBarLineLeft().getStyle();
String expectStyle1 = "heavy-light";
assertEquals(acturalStyle1,expectStyle1);

String actualRepeatType1 =
score.getParts().get(0).getMeasures().get(0).getBarLineLeft().getRepeatType();
String expectRepeatType1 = "forward";
assertEquals(actualRepeatType1,expectRepeatType1);

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

```
String acturalStyle2 =
score.getParts().get(0).getMeasures().get(0).getBarLineRight().getStyle();
String expectStyle2 = "light-heavy";
assertEquals(acturalStyle2,expectStyle2);

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

int actualRepeaNumber2 =
score.getParts().get(0).getMeasures().get(0).getBarLineRight().getRepeatNum();
int expectRepeaNumber2 = 8;
assertEquals(actualRepeaNumber2,expectRepeaNumber2);
```

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) {
```

```
                        existNotation = true;
                    }
                }
                Assertions.assertTrue(existNotation == false);
```

Output: True

Name: + notationTest4()
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/demoDrumsSimple2.musicxml");

```
                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);
```

Output: True

Name: + notationTest5()
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");

```
                int numOfNotaion = 0;
                List<Note> noteList1 =
score.getParts().get(0).getMeasures().get(1).getNotes();
                for(int i = 0; i < noteList1.size(); i++) {
                    if(noteList1.get(i).getNotation() != null) {
                        numOfNotaion++;
                    }
                }
                Assertions.assertTrue(numOfNotaion == 2);
                numOfNotaion = 0;

                List<Note> noteList2 =
score.getParts().get(0).getMeasures().get(0).getNotes();
                for(int i = 0; i < noteList2.size(); i++) {
                    if(noteList2.get(i).getNotation() != null) {
                        numOfNotaion++;
                    }
                }
                Assertions.assertTrue(numOfNotaion == 2);
```

Output: True

Name: + notationTestGetSlur()
Description: Test the getSlur() method in Notation.class
Input: setUp("src/test/resources/system/demoGuitarSimple4.musicxml");
List<Slur> list =
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getNotation().getSlurs();
Assertions.assertTrue(list.size() == 1);
Output: True

Name: +notationTestGetFret()
Description: Test the getFret() method in Notation.class
Input: 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);
Output: True

Name: + notationTestGetString()
Description: Test the getString() method in Notation.class
Input: 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);
Output: True

Name: + notationTestGetTech()
Description: Test the getTech() method in Notation.class
Input: 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"));

Technical tec2 =
score.getParts().get(0).getMeasures().get(1).getNotes().get(0).getNotation().getTechnical();
Assertions.assertEquals(tec2.getPullOff().get(0).getNumber(), 2);

Assertions.assertTrue(tec2.getPullOff().get(0).getType().equals("start"));

Assertions.assertTrue(tec2.getPullOff().get(0).getSymbol().equals("P"));

Technical tec3 =
score.getParts().get(0).getMeasures().get(17).getNotes().get(11).getNotation().getTechnical();
Assertions.assertEquals(tec3.getHammerOns().get(0).getNumber(),
2);

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().getTec
hnical().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().get
Ornaments();
        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:

Name: + testNoteNumber()
Description: Test the total number of note by using its list
Input: setUp("src/test/resources/system/demoDrumsSimple1.musicxml");
List<Note> noteList1 =
score.getParts().get(0).getMeasures().get(0).getNotes();
Assertions.assertTrue(noteList1.size() == 13);

Output: True

Name: + testGetNotaion()
Description: Test getNotation() method in Note.class
Input: setUp("src/test/resources/system/demoDrumsSimple2.musicxml");
List<Note> 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);
Output: True

Name: + testGetNoteType()
Description: Test getType() method in Note.class
Input: setUp("src/test/resources/system/demoDrumsSimple1.musicxml");
int notetype1 =
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getType();
Assertions.assertEquals(notetype1, 8);

int notetype2 =
score.getParts().get(0).getMeasures().get(0).getNotes().get(1).getType();
Assertions.assertEquals(notetype2, 8);

setUp("src/test/resources/system/demoGuitarSimple4.musicxml");

int notetype3 =
score.getParts().get(0).getMeasures().get(0).getNotes().get(0).getType();
Assertions.assertEquals(notetype3, 16);

int notetype4 =
score.getParts().get(0).getMeasures().get(0).getNotes().get(1).getType();
Assertions.assertEquals(notetype4, 8);
Output: True

Name: + testGetNotedot()

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();
```

```
                int expectNumberOfInstrument = 13;
                assertEquals(actualNumOfInstrument,expectNumberOfInstrument);
```
Output: True

Name: + TestgetMeasures()
Description: Test getMeasures() method in Part.class
Input: setUp("src/test/resources/system/demoDrumsComplex1.musicxml");

```
                int actualNumOfMeasures
=score.getParts().get(0).getMeasures().size();
                int expectNumberOfMeasures = 126;
                assertEquals(actualNumOfMeasures,expectNumberOfMeasures);
```
Output: 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 IndexOutOfBoundsExceptions (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 tieds 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. Test Coverage Analysis