TAB TO MUSICXML CONVERTER AND PLAYER

Testing Document

EECS 2311 -- Group 8

ATTRIBUTION

Parth Sharma

Mohammad Khan

Greatlove Bariboloka

Kedamawi Mengitsu

Sina Aligholizadeh

Winter 2022

PURPOSE

This document will describe the test cases written in JUnit framework for the MusicXML converter application.

INTRODUCTION

This section of the MusicXML converter application will convert MusicXML to sheet music and allow the user to play it.

PARSER

There are collections of JUnit tests that test for various aspects of the parser so each collection will have its own section.

Section 1: checks if parser can retrieve **instrument name** from xml string.

Section 2: checks if parser can retrieve the correct **number of measures** in the xml string.

Section 3: checks if parser can retrieve the correct **number of notes** in the cmlm string.

Section 3: checks if parser can preserve **all the information about a note** in the xml string

These are replicated for three instruments (guitar, drum set and bass) which makes up 12 test cases in total - a sufficient number of test cases for the size of this application.

Section 1

I. `test1_BendTestPartName()` for Guitar

II. `test2_PushTestPartName()` for Drumset

III. `test3_MoneyTestPartName()` for Bass

These test cases test the parser to see if it is able to retrieve the instrument name which, in the xml, is referred to as `partname`. The parser object (or "holder"), an object of type `ScorePartwise2`, is created by supplying the xml string corresponding to each of the test tabs (which are held and returned by helper methods in the `XmlToJavaTester` class) to the `unmarshal` method in the `XmlToJava` class.

These helper methods include `BendTestHolder()`, `PushTestHolder()`, `MoneyTestHolder()` as well the class of the parser object (`ScorePartwise2`). All of these are supplied to the `unmarshal` method. The string `partname` is extracted from the parser object via static parser methods belonging to a separate class `ListOfMeasureAndNote` which simplifies the extraction process. Then, an `assertTrue` method is used to compare the string that the partname holds to the string that represents the part name element in the xml string.

Section 2

I. `test1_BendTestNumofMeasures()` for Guitar

II. `test2_PushTestNumofMeasures()` for Drumset

III. `test3_MoneyTestNumofMeasures()` for Bass

These test cases check that the parser is able to extract the total number of measures from the xml string. A `ScorePartwise2` object is instantiated from the `unmarshal` method which does the conversion of the xml string to the connected tree of objects. The xml string is supplied to this `unmarshal` method as well.

An `assertTrue` method is used to see if the extracted value for the number of measures is equivalent to the expected value which is derived directly from the xml string.

Section 3

I. `test1_BendTestNumofNotes()` for Guitar

II. `test2_PushTestNumofNotes()` for Drumset

III. `test3_MoneyTestNumofNotes()` for Bass

These test cases check to see if the parser object is able to store the correct information about the number of notes in the xml string. The parser extracts the number of notes by adding the number under each measure. An `assertTrue` method is used to compare the extracted value of the number of notes to the number representing the amount of notes that the xml string has.

Section 4

I. `test1_BendTestGetFirstNoteInfo()` for Guitar

II. `test2_PushTestGetFirstNoteInfo()` for Drumset

III. `test3_MoneyTestGetFirstNoteInfo()` for Bass

These test cases check to see if the parser object stores the correct note, the correct information about that note, and has access to both. Similarly, an `assertTrue` is used to determine whether the `display` information (`frets` for guitar and base, and `noteheads` for drum sets) of the first note in each of the xml strings is the same as the `display` information stored by the parser object.

GUI

A class named `DrawLine` was added to the `src/main/GUI` package and its purpose is to handle the display of lines while previewing sheet music with regards to the music xml string information. The tests for this class include:

I. `testGetLine()`

II. `testSetLine()`

`testGetline()`

Tests that the coordinates (of data type `double`) provided are the same coordinates returned from the `getLine()` method in the DrawLine class.

First, a line object of the `DrawLine` class is initialized with x and y coordinates. Afterwards, each coordinate is independently compared with the x and y coordinates retrieved from the getter method of the `DrawLine` class.

Only this test case is necessary to test the `getLine()` method because if the assertion is true for one `DrawLine` object then it will be true for any other `DrawLine` object, because the only difference between lines is their x and y coordinates (which is already being tested).

`testSetline()`

Tests that the coordinates (of data type `double`) are the same coordinates returned from the `getLine()` method after these coordinates have been changed through the `setLine()` method.

First, an instance of the `DrawLine` class called `l1` is initialized with a set of x and y coordinates. Then, another instance called `l2` is initialized with different coordinates. The coordinates of `l1` are then changed to those of `l2` using the `setLine(DrawLine l)` method, with `l2` being passed as a parameter. The assumption is that the coordinates of `l1` have changed to those of `l2`, hence and assertion is done to confirm this. If the test is successful then the coordinates were changed correctly.

Only this test case is necessary to test the `setLine()` method because if the assertion is true for one `DrawLine` object, then it will be true for any other `DrawLine` object, because the only difference between lines is their x and y coordinates.