TAB2MXL: Txt to MusicXML File Converter

Testing Document

Sunday February 28, 2021

EECS 2032 - Professor Tzerpos

Group 12
Sara Araibi (215700255)
Vishwa Perera (216155947)
Savneet Gill (217386400)
Nishiket Singh (216521197)
Kaneez Fatima (215711534)

Table of Contents

1.0 Test Case Analysis ——— 3

1.1 Application and Software System Overview ——— 3
1.2 Test Case Scope ——— 3

2.0 Test Case Design ——— 3

3.0 Test Case Description ——— 4

4.0 Testing Requirements and Features ——— 11

1.0 Test Case Analysis

1.1 Application and Software System Overview

The results of the test cases reflect on our software system and application of the project. The TAB2MXL takes a music tablature in the format of a text file and converts it to an MusicXML file. Our application is a Java application where it will be running through a gradle task in eclipse. It has features including where the application can save the file, add the title and so on.

1.2 Test Case Scope

The scope of our test cases defines on what method the test case is testing. There are different functions that are used in the scope of the test case. The test cases demonstrated the different methods that is being tested for the product of the customer. These areas that have been tested are a crucial part of the software system as to it shows the customer that the software system is being run to the expectation of the back-end user. Majority of the methods written in the project are ensured that it has been tested to ensure that the method can read the octave, step, alter and so on. These are crucial part of the project for the tablature of the guitar and bass instrument.

2.0 Test case Design

When designing test cases, there are various inputs and expected outputs in relation to our project. The test case design is using JUnit library in part of the testing design. The inputs in some of our test cases are designs that it is comparing the expected values to the text file being called. Such inputs help ensure that we can compare the output of the text file to the designated output of what the user needs. Test cases include different kind of functions that we are using such as the assertEquals, scanners, assertTrue and so on. These functions are essential part of the test case implementation as to it help us compare, checks if the output is true and so on. Please note that the software system of the drum set is not yet to be completed and is in progress.

3.0 Test Case Description

TestPitch.java					
Test Case Names	Expected	Actual	Results (Pass/Fail)	How it was derived	How it was implemented
testStep_Normal	"C"	"C"	PASS	The test case was derived to check to see if the expected output which is the character, "C", to the actual output of the variable p. The variable p stores the new note.	It was implemented by using String called expected where it stored the expected character. The Pitch class is getting called by the variable p where it is storing a new note. The expected variable and the p.getStep() is then being compared by using the function assertEquals.
testStep_Sharp	"C"	"C"	PASS	The test case was derived to check to see if the expected output which is the character, "C", to the actual output of the variable p. The variable p stores the new note.	It was implemented by using String called expected where it stored the expected character. The Pitch class is getting called by the variable p

					where it is storing a new note. The expected variable and the p.getStep() is then being compared by using the function assertEquals.
testAlter_Normal	0	0	PASS	The test case was derived to check to see if the expected output which is the numeric value ,0, to the actual output of the variable p. The variable p stores the new note.	It was implemented by using int to store the expected value in this variable. The Pitch class is getting called by the variable p where it is storing a new note. The expected variable and the p.getAlter() is then being compared by using the function assertEquals.
testAlter_Sharp	1	1	PASS	The test case was derived to check to see if the expected output which is the numeric value ,0, to the actual output of the variable p. The variable p stores the new note.	It was implemented by using int to store the expected value in this variable. The Pitch class is getting called by the variable p where it is

					storing a new note. The expected variable and the p.getAlter() is then being compared by using the function assertEquals.
testOctave_Normal	5	5	PASS	The test case was derived to check to see if the expected output which is the numeric value ,5, to the actual output of the variable p. The variable p stores the new note.	It was implemented by using int to store the expected value in this variable. The Pitch class is getting called by the variable p where it is storing a new note. The expected variable and the p.getOctaver() is then being compared by using the function assertEquals.
testOctave_Sharp	5	5	PASS	The test case was derived to check to see if the expected output which is the numeric value ,5, to the actual output of the variable p. The variable p stores the new note.	It was implemented by using int to store the expected value in this variable. The Pitch class is getting called by the variable p where it is storing a new note. The

					expected variable and the p.getOctave() is then being compared by using the function assertEquals.
testEquals	True False	True False	PASS	To check the equality between two pitches.	Creating two pitch objects with the same/different step, alter, and octave. Checks the equality by calling the equals method.

Why is it sufficient?

These test cases are sufficient to use because when writing out methods we wanted to check if the calculations for the alter, step, and octave are correct. These methods are important in terms of the MusicXML file output to get the correct values or characters when inputting a text file.

TestTabReader.java					
Test Case Names	Expected	Actual	Results (Pass/Fail)	Why/How it was derived	How it was implemented/
testReadFile	Reads the contents of test_tabs_reading.txt	Reads the contents of test_tabs_reading.txt	PASS	We used a standard text tab.	Each line in the file was compared to the expected list of strings.
testSplitMeasure	Returns a list of measure as a list of strings.	Returns a list of measure as a list of strings.	PASS	We wanted to split the measure wherever there is a line of vertical bars.	The measures returned by splitMeasure is compared against the expected measures.

testLineHasTabs	Returns true when the line contains at least 2 vertical bars and 2 dashes.	Returns true when the line contains at least 2 vertical bars and 2 dashes.	PASS	Testing to check if a string has 2 vertical bars and 2 dashes.	1.Returns true if the string has 2 vertical bars and 2 dashes 2. Returns false if the string has 1 vertical bar and 2 dashes 3. Returns false if the string has 2 vertical bars and 1 dash
testGetTuning	[E,B,G,D,A,E]	[E,B,G,D,A,E]	PASS	Testing standard tuning.	Adding the standard tuning to a list and comparing it to getTuning().
testGetTitle	Gets the file's name, test_tabs_reading	Gets the file's name, test_tabs_reading	PASS	To get the title of the text file.	The value returned by getTitle to test_tabs_reading
testGetInstrument	Returns the instrument whether its "Classical Guitar" or "Drumset".	Returns the instrument whether its "Classical Guitar" or "Drumset".	PASS	Reading two different text tabs, one for guitar and one for drums	The value returned by getInstrument to the designated text file's instrument.

Why is it sufficient?

These test cases are sufficient because it checks if the method is producing the right output or that it ensures that the method we have written is being used as the user expected. For example, the test case of the testReadFile checks whether or not an input text file can be read from software system.

TestNote.java					
Test Case Names	Expected	Actual	Results (Pass/Fail)	How it was derived?	How it was implemented ?
testCompareTo_Negative	True	True	PASS	True when this note precedes	This was compared by using

				the other note when being compared.	assertTrue to check if n1 and n2 is less than 0.
testCompareTo_Positive	True	True	PASS	True when the other note precedes this note when being compared.	This was compared by using assertTrue to check if n1 and n2 is greater than 0.
testCompareTo_Equals	True	True	PASS	True when this note is either above or below the other note.	This was compared by using assertTrue to check if n1 and n2 is equal to 0.

Why is it sufficient?

These test cases are sufficient to use because in the class note there are codes written for the guitar/bass in terms of notes. There are also codes written for the drum instrument in terms of notes. (Still in process of completing). These codes need to ensure that it is reading the note of the desired instrument accordingly. If a note is not being expected to the user needs (which would be us) it can help determine where the problem is. For example, it can help the user to check if it is giving the desired tuning or fret number and so on.

TestMeasure.java					
Test Case Names	Expected	Actual	Results (Pass/Fail)	How it was derived?	How it was implemented?
testMeasure	Constructor successful	Constructor successful	PASS	To check whether the constructor does not crash.	This was implemented using the try and catch block.
testSortArray	Sorts the notes by non-descending order.t	Sorts the notes by non-descending order.	PASS	This was derived by testing to see if the measures are in ascending order.	This was implemented by taking a text file and looping through the measures to put them in

					ascending order.

Why is it sufficient?

These test cases are sufficient to our software system because it checks whether the measures are in ascending order in order to produce the designated MusicXML file. This is sufficient to the software system as to it determines the order of the measures and how it would be disignated on the designated XML file.

TestAttributes.java					
Test Case Names	Expected	Actual	Results (Pass/Fail)	How it was derived?	How it was implemented?
testAttributes	Constructor successful	Constructor successful	PASS	To check whether the constructor does not crash.	It was implemented using the try and catch block.

Why is it sufficient?

These test cases above are sufficient to our software system because in a guitar tab or bass tablature there are different tuning. With this said, the different tuning defines the different pitches. However, it is crucial that the testAttributes count for all of this. As a result of this, coding these test cases aids in the process to check whether the guitar tuning has been done properly or if there are any error that as a user need to take care off.

4.0 Testing Requirement and Features

The testing requirements for each test case is using the JUnit 5 library that needs to be installed in order to run the test cases. This is an essential part of the written test cases because without it the test cases will not run. When writing the test case for our methods, the proper syntax at the beginning is to write *@Test* to ensure that the software will know that this is a test case method.

The features that the test case are testing are the guitar instrument tablatures of the guitar and the bass instrument tablatures.