TablatureConverter

Testing Document

EECS 2311 – Group 6

Tuan Dau

James Le

Temiloluwa Odunfa

Mohammad Amin Mohammadi

02/28/2021

Table of Contents

1.Introduction

The purpose of this document is to describe all the test cases run in the Tablature Converter program, the way they were derived and the way they were implemented. It talks in detail why the front-end and back-end testings are sufficient enough for a fully functioning program.

2.Testing Methodology

Test cases are implemented in the TablatureConverter project in order to ensure smoothness in the user experience, as well as to streamline development by reducing the likelihood of encountering code faults later in development. All of the unit tests conducted in the TablatureConverter project are written in JUnit 5.

3.Front-End Tests

3.1 GUI Testing

3.1.1 Main Test

Test Case1: containsUploadFileButton(FxRobot robot)

Description:This method verifies that the UploadFileButton has the correct text.

Test Steps:

1. Create a **start** method that brings up the main scene(GUI) before you start testing
2. Test that the #id associated with the UploadFileButton has the correct text.

Test Case2: containsConvertButton(FxRobot robot)

Description:This method verifies that the ConvertButton has the correct text.

Test Steps:

1. Create a **start** method that brings up the main scene(GUI) before you start testing
2. Test that the #id associated with the ConvertButton has the correct text.

Test Case3: containsSaveButton(FxRobot robot)

Description:This method verifies that the SaveButton has the correct text.

Test Steps:

1. Create a **start** method that brings up the main scene(GUI) before you start testing
2. Test that the #id associated with the SaveButton has the correct text.

Test Case4: codeAreaTest(FxRobot robot)

Description:This method checks if the "codeArea" gets the correct input from the user

Test Steps:

1. Use the FxRobot function to click on the codeArea
2. Check using the assertEquals function that the codeArea accepts the correct text imputes.

Test Case5: songTitleTest(FxRobot robot)

Description:This method checks if the "title" gets the correct input from the user

Test Steps:

1. Use the FxRobot to click on the title field and write a text in the code area
2. Check using the assertEquals function that the title field accepts the correct text format.

Test Case6: measureTest(FxRobot robot)

Description:This method tests the go-to-measure function, it uses the FxRobot to click on the "codeArea" and write a tablature in order to check if the measure number is correct

Test Steps:

1. Create an integer for a measure number
2. Use the FxRobot function to click on the "codeArea" and write a tablature
3. Use the FxRobot function to click on the Go-To-Measure textfield and write a number to go to.
4. Check using the assertEquals function that the imputed measure number corresponds to the specific part in the tablature.

Test Case7: convertTest(FxRobot robot)

Description:This method tests the go-to-measure function, it uses the FxRobot to click on the "codeArea" and write a tablature in order to check if the measure number is correct

Test Steps:

Test Case8: convertTest2(FxRobot robot)

Description:This method checks if the application changes view to the output screen when the ConvertButton is clicked

Test Steps:

1. Use the FxRobot function to click on the "codeArea"
2. Use the FxRobot function to write a correct tablature format
3. Use the FxRobot function to click on the "ConvertButton"
4. Check is **true** when the assertEquals function confirms that the output screen is displayed.

Test Case9: XMLViewerTest(FxRobot robot)

Description: This method checks that the XMLTextArea isn't empty when a tablature file has been converted

Test Steps:

1. Use the FxRobot function to click on the "codeArea"
2. Use the FxRobot function to write a correct tablature format
3. Use the FxRobot function to click on the "ConvertButton"
4. Check is **true** when the assertEquals function confirms that the XMLTextArea isn't empty
5. Close current window using the FxRobot function

Test Case10: timeSignatureTest(FxRobot robot)

Description: This method checks the "timeSignatureList" to see if it is the right one selected one by the user

Test Steps:

1. Use the FxRobot function to click on the "timeSignatureList"
2. Check using the assertEquals function that the "timeSignatureList"displays the selected one by user.

Test Case11: convertButtonColorTest(FxRobot robot)

Description: This method checks if the color of the "Convert" button is blue when the "codeArea" isn't empty.

Test Steps:

1. Use the FxRobot function to click on the "codeArea"
2. Use the FxRobot function to write anything
3. Check using the assertEquals function that the color of the "Convert" button is blue.

Test Case12: incorrectTuningErrorTest(FxRobot robot)

Description:This method checks if the correct error message is displayed when there is an incorrect tuning in the tablature file.

Test Steps:

1. Use the FxRobot function to click on the “codeArea”
2. Use the FxRobot function to write a tablature with incorrect tuning
3. Use the FxRobot function to click on the “convertButton”
4. Check using the assertEquals function that the correct error message is displayed.
5. Close current window using the FxRobot function

Test Case13: misalignedTablatureErrorTest(FxRobot robot)

Description:This method checks if the correct error message is displayed when the tablature file is misaligned during the conversion process.

Test Steps:

1. Use the FxRobot function to click on the “codeArea”
2. Use the FxRobot function to write a tablature in a misaligned format
3. Use the FxRobot function to click on the “convertButton”
4. Check using the assertEquals function that the correct error message is displayed.
5. Close current window using the FxRobot function

Test Case14: invalidTabErrorTest(FxRobot robot)

Description: This method checks if the correct error message is displayed when there is an invalid tab to be converted.

Test Steps:

1. Use the FxRobot function to click on the “codeArea”
2. Use the FxRobot function to write an invalid text
3. Use the FxRobot function to click on the “convertButton”
4. Check using the assertEquals function that the correct error message is displayed.
5. Close current window using the FxRobot function

Test Case15: saveFileTest(FxRobot robot)

Description:This method verifies that the “Save” button stores the converted XML file in the system.

Test Steps:

1. Use the FxRobot function to click on the “outputTab”
2. Use the FxRobot function to click on the “SaveButton”
3. Use the FxRobot function to close the current window
4. Check using the assertEquals function that the converted XML file got saved.

Test Case16: uploadFileTest(FxRobot robot)

Description:This method checks if a file is created when a tablature is in the “codeArea”

Test Steps:

1. Use the FxRobot function to click on the “codeArea”
2. Use the FxRobot function to write a text”
3. Check **true** when the assertEquals function confirms that a file has been created.

4.Back-End Tests

4.1.Output Test

4.1.1 testOutput():

Description: This method tests that all constructors of the Output class can be run correctly. This test checks that the outputted object files against the expected fields that it should have for a given constructor run.

Test Steps:

1. Define a variable of Output by constructor of Output
2. Check if it stores it perfectly

4.1.2 testGetletter():

Description: This method tests that the created Output objects return the correct step (note value) stored in each Output object.

Test Steps:

1. Define a variable of Output by constructor of Output
2. Check if Getletter return the correct value

4.1.3 testGetnote1():

Description: This method tests that the created Output objects return the correct note1 field and that the return type is of the correct return type.

Test Steps:

1. Define a variable of Output by constructor of Output
2. Check if Getnote1 return the correct value

4.1.4 testGetnote2():

Description: This method tests that the created Output objects return the correct note2 field and that the return type is of the correct return type.

Test Steps:

1. Define a variable of Output by constructor of Output
2. Check if Getnote2 return the correct value

4.1.5 testGetindex():

Description: This method tests that the created Output objects return the correct index field and that the return type is of the correct return type.

Test Steps:

1. Define a variable of Output by constructor of Output
2. Check if Getindex return the correct value

4.1.6 testGetTech():

Description: This method tests that the created Output objects return the correct technique field and that the return type is of the correct return type.

Test Steps:

1. Define a variable of Output by constructor of Output
2. Check if GetTech return the correct value

4.2 TextReader Test

4.2.1 testParsGuitar():

Description: This method tests that bass tablature is being parsed properly by comparing the ParsGuitar()'s output to a manually generated string of what the team's expectations are for the given test tablature input.

Test Steps:

1. Define list of output and linkedHashMap of tab
2. Put each line of tab into the linkedHashMap one by one
3. Call ParsGuitar method to parse the tab
4. Check if the returned output by ParsGuitar is correct

4.2.2 testParsBass():

Description: This method tests that bass tablature is being parsed properly by comparing the ParsBass()'s output to a manually generated string of what the team's expectations are for the given test tablature input.

Test Steps:

1. Define list of output and linkedHashMap of tab
2. Put each line of tab into the linkedHashMap one by one
3. Call ParsBass method to parse the tab
4. Check if the returned output by ParsBass is correct

4.2.3 testParsDrum():

Description: This method tests that the bass tablature is being parsed properly by comparing the ParsDrum()'s output to a manually generated string of what the team's expectations are for the given test tablature input.

Test Steps:

1. Define list of output and linkedHashMap of tab
2. Put each line of tab into the linkedHashMap one by one
3. Call ParsDrum method to parse the tab
4. Check if the returned output by ParsDrum is correct

4.2.4 testisInteger():

Description: If the string given is an integer. A string is defined to be a valid integer when there are only numeric characters in the string, there are no decimal places, and that the represented number value in the string is less than (2^31)-1.

Test Steps:

1. Define a String variable which is a number
2. Check if isInteger return the correct boolean

4.2.5 testTabIsOK():

Description: This method tests to see if the format of the inputted tablature is valid, as defined by the rules in Section 3.1, item 3 of the Group 6 TablatureConverter Requirements Document.

Test Steps:

1. Define list of String of tab
2. Put each line of tab into the linkedHashMap one by one
3. Call TabIsOK to check if the tab is valid

4.2.6 testgetCharFromString():

Description: This method tests to see if a character can be taken from a string, given a string input, and an index given as an integer. The return boolean of getCharFromString() returns, and false if an indexOutOfBoundsException() is detected, and true if there is no indexOutOfBoundsException() exception. The test passes if the method returns the expected return boolean, and returns false otherwise.

Test Steps:

4.3 ConvertedSongTest Tests

4.3.1 testGetLastNote1():

Description: This method checks to see if one can retrieve the last note added to a ConvertedSong object. The “last note” is defined as the note that is in the last part, last measure, and last index of the List<Note> structure in the ConvertedSong object.

Test Steps:

1. Create a ConvertedSong object, and initialize it to have one part and measure.

2. Add a singular dummy note to the ConvertedSong object.
3. Retrieve the dummy note with the getLastNote() function.
4. Assert that the dummy note and the retrieved object are the same.

4.3.2 testGetLastNote2():

Description: This method checks to see if one can retrieve the last note added to a ConvertedSong object. This method differs from the test done in 4.3.1 as there are multiple notes added in this test.

Test Steps:

1. Create a ConvertedSong object, and initialize it to have one part and measure.
2. Add five dummy notes to the ConvertedSong object.
3. Retrieve the last dummy note with the getLastNote() function.
4. Assert that the last dummy note and the retrieved object are the same.

4.3.3 testGetLastNote3():

Description: This method checks to see if one can retrieve the last note added to a ConvertedSong object. This method differs from the test done in 4.3.1 and 4.3.2 as there are many more notes added in this test.

Test Steps:

1. Create a ConvertedSong object, and initialize it to have one part and measure.
2. Add twenty dummy notes to the ConvertedSong object.
3. Retrieve the last dummy note with the getLastNote() function.
4. Assert that the last dummy note and the retrieved object are the same.

4.3.4 testNoteType1():

Description: This method, given the number of dashes that a note takes up, as well as the divisions attributes of a song, will determine the type of note (whole, half, quarter, etc) that a note is.

Test Steps:

1. In a for loop, check every possible type that a note can be (all note types from ‘whole’ down to ‘128th’ note).
2. Run the noteType method for every value passed to the for loop, effectively testing all possible outputs of the noteType function. Assert that the actual type produced by the method is the same as the expected type produced manually by the tester.

4.3.5 testNoteType2 ():

Description: This method, given the number of dashes that a note takes up, as well as the total number of dashes in a measure, will determine the type of note (whole, half, quarter, etc) that a note is.

Test Steps:

1. In a for loop, check every possible type that a note can be (all note types from ‘whole’ down to ‘128th’ note).
2. Run the noteType method for every value passed to the for loop, effectively testing all possible outputs of the noteType function. Assert that the actual type produced by the method is the same as the expected type produced manually by the tester.

4.3.6 testCreateXML1 ():

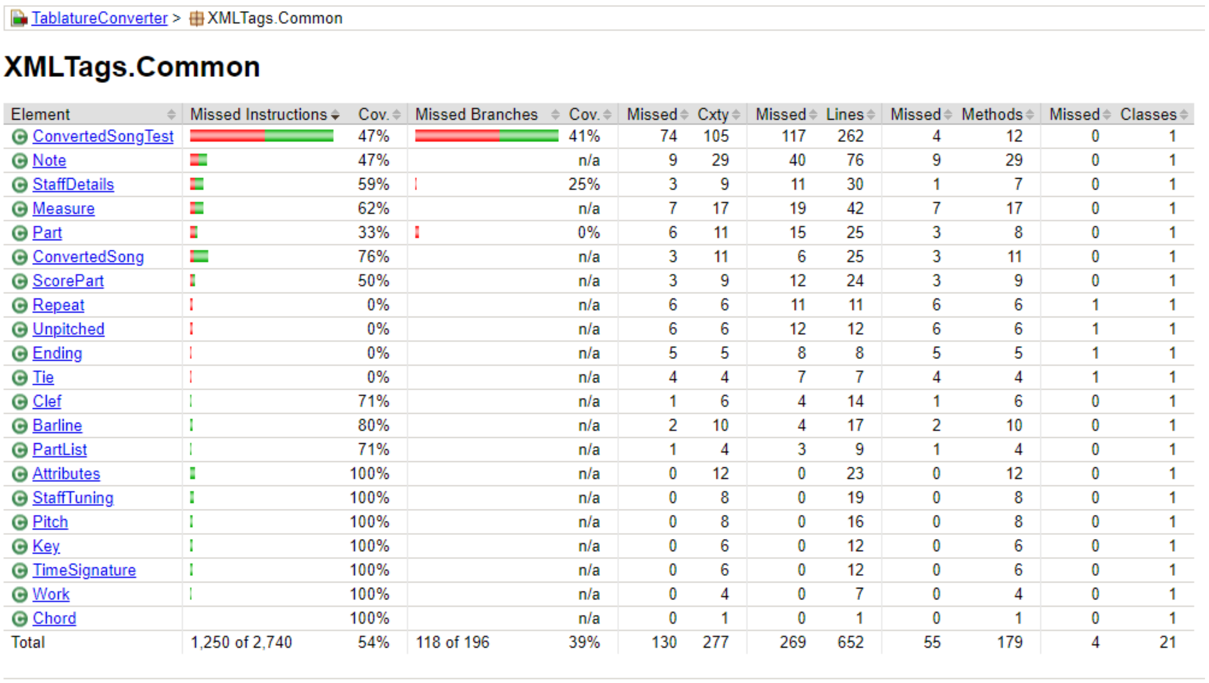
Description: This method is tested by passing an example tablature, which in this case, is hot cross buns, and compares it to a manually generated MusicXML file prepared by the tester.

Test Steps:

1. Declare the input, output, and expected output file paths.
2. Create an XML file with the ConvertedSongTest.createXML method, giving the input file of Hot Cross Buns, and an output file path to a local project directory.
3. The actual and expected output files are read line by line and then read as bytes, and compared by each byte. The test passes if every byte read from each file is exactly the same as the other.

5.Testing Sufficiency

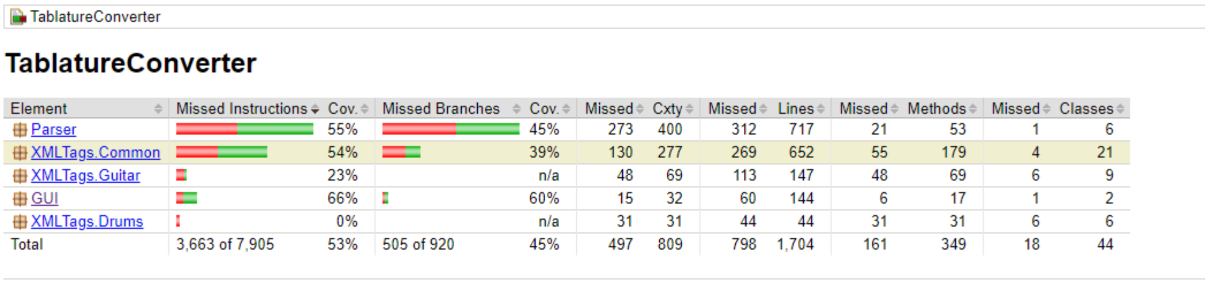
All test cases are included by making one test class for every class that needs to be tested, such as Output and textReader. Each method has one test case associated with it to ensure reliability. Attached below are the coverage for the testing of the Tablature Converter.



TablatureConverter > XMLTags.Common

XMLTags.Common

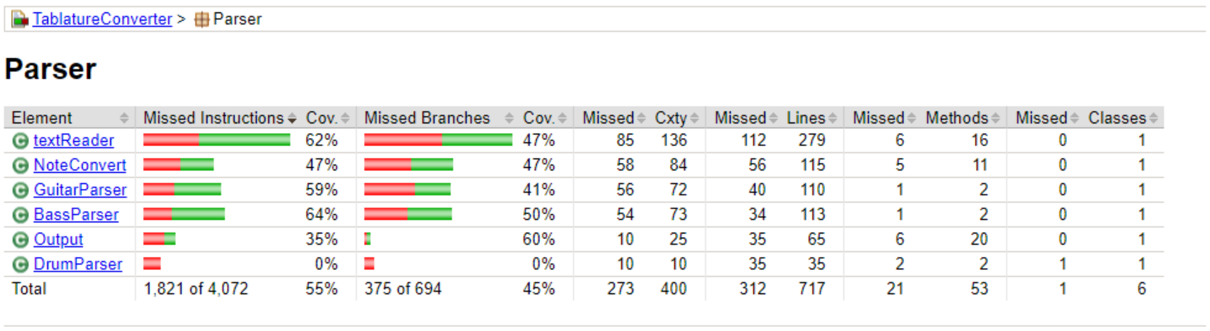
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
ConvertedSongTest		47%		41%	74	105	117	262	4	12	0	1
Note		47%		n/a	9	29	40	76	9	29	0	1
StaffDetails		59%		25%	3	9	11	30	1	7	0	1
Measure		62%		n/a	7	17	19	42	7	17	0	1
Part		33%		0%	6	11	15	25	3	8	0	1
ConvertedSong		76%		n/a	3	11	6	25	3	11	0	1
ScorePart		50%		n/a	3	9	12	24	3	9	0	1
Repeat		0%		n/a	6	6	11	11	6	6	1	1
Unpitched		0%		n/a	6	6	12	12	6	6	1	1
Ending		0%		n/a	5	5	8	8	5	5	1	1
Tie		0%		n/a	4	4	7	7	4	4	1	1
Clef		71%		n/a	1	6	4	14	1	6	0	1
Barline		80%		n/a	2	10	4	17	2	10	0	1
PartList		71%		n/a	1	4	3	9	1	4	0	1
Attributes		100%		n/a	0	12	0	23	0	12	0	1
StaffTuning		100%		n/a	0	8	0	19	0	8	0	1
Pitch		100%		n/a	0	8	0	16	0	8	0	1
Key		100%		n/a	0	6	0	12	0	6	0	1
TimeSignature		100%		n/a	0	6	0	12	0	6	0	1
Work		100%		n/a	0	4	0	7	0	4	0	1
Chord		100%		n/a	0	1	0	1	0	1	0	1
Total	1,250 of 2,740	54%	118 of 196	39%	130	277	269	652	55	179	4	21



TablatureConverter

TablatureConverter

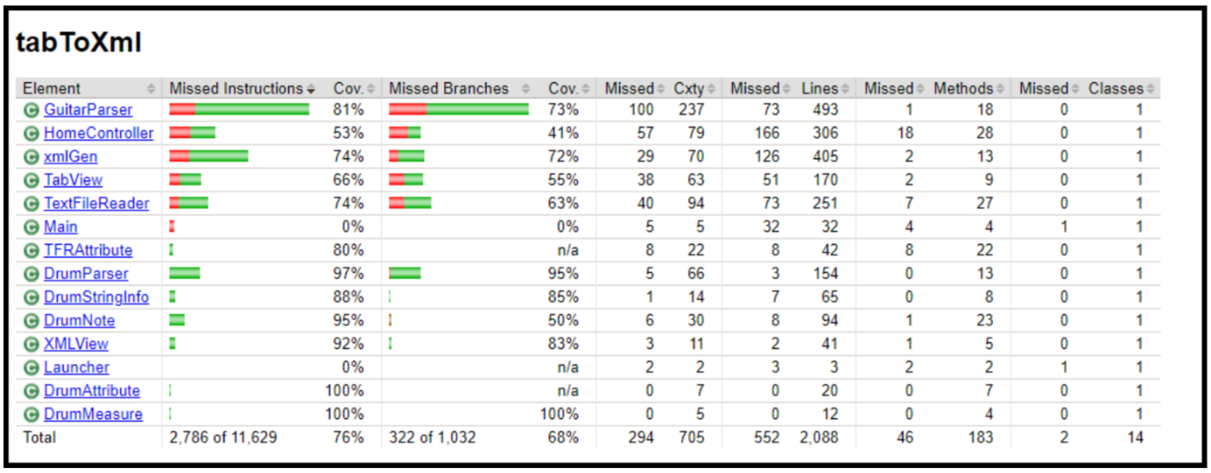
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Parser		55%		45%	273	400	312	717	21	53	1	6
XMLTags.Common		54%		39%	130	277	269	652	55	179	4	21
XMLTags.Guitar		23%		n/a	48	69	113	147	48	69	6	9
GUI		66%		60%	15	32	60	144	6	17	1	2
XMLTags.Drums		0%		n/a	31	31	44	44	31	31	6	6
Total	3,663 of 7,905	53%	505 of 920	45%	497	809	798	1,704	161	349	18	44



TablatureConverter > Parser

Parser

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
textReader		62%		47%	85	136	112	279	6	16	0	1
NoteConvert		47%		47%	58	84	56	115	5	11	0	1
GuitarParser		59%		41%	56	72	40	110	1	2	0	1
BassParser		64%		50%	54	73	34	113	1	2	0	1
Output		35%		60%	10	25	35	65	6	20	0	1
DrumParser		0%		0%	10	10	35	35	2	2	1	1
Total	1,821 of 4,072	55%	375 of 694	45%	273	400	312	717	21	53	1	6



tabToXml

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
GuitarParser		81%		73%	100	237	73	493	1	18	0	1
HomeController		53%		41%	57	79	166	306	18	28	0	1
xmlGen		74%		72%	29	70	126	405	2	13	0	1
TabView		66%		55%	38	63	51	170	2	9	0	1
TextFileReader		74%		63%	40	94	73	251	7	27	0	1
Main		0%		0%	5	5	32	32	4	4	1	1
TFRAttribute		80%		n/a	8	22	8	42	8	22	0	1
DrumParser		97%		95%	5	66	3	154	0	13	0	1
DrumStringInfo		88%		85%	1	14	7	65	0	8	0	1
DrumNote		95%		50%	6	30	8	94	1	23	0	1
XMLView		92%		83%	3	11	2	41	1	5	0	1
Launcher		0%		n/a	2	2	3	3	2	2	1	1
DrumAttribute		100%		n/a	0	7	0	20	0	7	0	1
DrumMeasure		100%		100%	0	5	0	12	0	4	0	1
Total	2,786 of 11,629	76%	322 of 1,032	68%	294	705	552	2,088	46	183	2	14

The testing is sufficient as it ensures the basic functionality of each method, which can be used to build the overarching system. The tests above cover the majority of the system. This shows that the most important part of our system has been tested thoroughly.