Testing Document

ℹ️ **EECS 2311 W22 - Group 9** `v0.2`

Members:
- Vivek Wadhwani (*vivek121@my.yorku.ca*)
- Kris Singh (*ksingh7@my.yorku.ca*)
- Kingsley Okon (*King808@my.yorku.ca*)
- Lan Zhang (*zhalala8@my.yorku.ca*)

Project: github.com/devivekw/TAB2XML

Documentation Link

Table of Contents

1 Introduction to Testing

Test-driven development is a core concept of programming, especially in a team. By creating a series of test cases and working to pass them, it can be trusted that the given program will work as intended when a user attempts to use the application by themselves. To make these tests, the JUnit testing framework was used with its relatively simple uses and efficient applications.

1.1 Testing a JavaFX Application

This project uses JavaFX to display the GUI to the user. JavaFX helps construct the different windows within the application, namely the implementations for the Play Music and Preview Sheet Music functions. Since JavaFX applications contain the instance information within the actual JavaFX windows, the process for grabbing the needed information is different from simply calling Java methods normally. As such, each of the 2 test classes created for this project have a start method that initializes the JavaFX menu.

1.2 Using FxRobot for Testing

To combat the issue of testing the aforementioned JavaFX application, the FxRobot library was used to assist in the tests. FxRobot essentially allows commands to be preprogrammed such that buttons on the application can be clicked and text can be entered without any human interference. These robot interactions within the application in addition to the FxAssert library allows the test cases to enter and collect the required information for testing.

1.3 Sample Tablature

To keep the testing consistent, the same tablature was used for each test case that required an input. The sample tablature used was the following:

```
CC|x---------------|--------x-------|
HH|--x-x-x-x-x-x-x-|----------------|
SD|----o-------o---|oooo------------|
HT|----------------|----oo----------|
MT|----------------|------oo--------|
BD|o-------o-------|o-------o-------|
```

2 Testing the Playing Functionality

The playing functionality allows the user to input guitar or drum tablature into the application and play the notes. For the following subheadings, they will be named after the test cases that were run.

2.1 testAssertions

This test case was created initially to get comfortable with the testing framework and simply makes sure that the assertion commands work as intended.

2.2 testNoInput

This test case launches the main view and attempts to click the Play Music button. It tests the GUI element of the Play Button to make sure that it is disabled if there is no valid text.

2.3 testWithInvalidInput

Similar to testNoInput, this test case tests the Play Music button such that it is disabled if there is invalid text. This is significant as there is text unlike the previous test case, but it is not valid drum or guitar tablature.

2.4 testWithValidInput

The final of the testing input cases for the playing functionality, this test case inputs a valid tablature as seen in 1.3 and checks to see if the Play Music button is enabled.

2.5 testPlayPause

This test case is important as it tests the main playing and pausing functionality. After some consideration, it was determined that one efficient way to test this was to test the application with the given tablature and check the current time on the time tracking slider at several point. To accomplish this, FxRobot was used to implement the sample tablature and navigate to the correct menus. After arriving at the Play Music window, the current time is immediately checked which should be 00:00 as the window has only just launched. The play and pause buttons are also tested to make sure they are enabled. Then, the robot will press the play button for one second and then pause. The current time will be checked again and the expected value is not zero as some time has elapsed.

2.6 testBPM

One unique feature present in the application is the capability for the user to change the bpm(beats per minute) of the playing. One part of this feature is that whenever the bpm is changed, the end time of the time tracking slider would also change to account for the difference in bpm. As such, the way in which it was determined to test this feature was to look at the end time before and after updating the bpm using the provided bpm slider. To go about this, FxRobot was used in a similar manner as in the last test case; it enters the sample input and navigates to the Play Music menu. Here it first determines if the bpm slider is at its default value of 120.0. It plays and pauses the music to initialize the window and then the ending time would be checked and compared to the predetermined value for the default bpm. Then, the robot will move the bpm slider to the right and then test to determine if the ending time has changed from the default which it should have, concluding the test.

2.7 Discussion

Altogether, the testing for the playing function was fairly thorough. The first 4 test cases were more minor than the latter 2, but they were important in determining if the GUI was working as intended. The important features like playing and pausing were tested and worked as intended. Considering the functionality for those 2 buttons is fairly straightforward, further testing would not be required. Next, the bpm feature was tested diligently through the use of the robot moving the slider and testing for a change in the end time. Since all that was talked about here were the major functionalities, this should be a sufficient amount of testing.

3 Testing the Sheet Music Functionality

The other major component of this project is the ability to display the sheet music for the provided tablature.

3.1 testNoInput

This test case launches the main view and attempts to click the Preview Sheet Music button. It tests the GUI element of the Preview Sheet Music button to make sure that it is disabled if there is no valid text.

3.2 testWithInvalidInput

Similar to the last test, this test case tests the Preview Sheet Music button such that it is disabled if there is invalid text. This is significant as there is text unlike the previous test case, but it is not valid drum or guitar tablature.

3.3 testWithValidInput

The final of the testing input cases for the playing functionality, this test case inputs a valid tablature as seen in 1.3 and checks to see if the Preview Sheet Music button is enabled.

3.4 testNoteToNumber

This test case is meant to test the important method NoteToNumber in the PrevSheetMusicController class. The tested method is what determines the notes and their place on the staff. It was actually fairly easy to test this as it simply needs sample inputs. Thus, this test case runs the method with a couple of different inputs and asserts that they are equal to the expected output.

3.5 testWithSlur

The final test for the sheet music output, this test is meant to ensure the slurs created by pull-offs work as intended. This was derived after the implementation of the slur feature was added to make sure the code runs. Although the test itself does not check if

the slur exists, by looking at the code coverage document, it can be seen that the code was successfully run and the desired output would be obtained.

3.6 Discussion

The implementation of the sheet music function was more straightforward than the diverse array of features in the play music functionality which made it altogether easier to write tests for. The first 3 tests were similar to 2.2 - 2.4 from the play music tests, and were merely to make sure the GUI was working as intended. As for determining the notes, there is one method that handles converting the notes given in the music xml, and so by testing a couple of input against that method and passing, it can be determined that the information is being correctly parsed from the xml that is being generated from the tablature. As such, this should be sufficient testing for this functionality.

4 Coverage

The following coverage document was generated from our tests:

TAB2XML_G14

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
GUI		25%		13%	203	242	628	818	107	139	15	22
converter.note		45%		39%	192	299	329	687	61	125	4	14
converter.instruction		36%		27%	78	101	139	222	16	35	1	6
converter		74%		67%	62	153	89	431	6	49	0	7
utility		84%		69%	70	186	105	543	17	81	1	13
draw.score		85%		68%	41	113	71	467	3	38	0	6
converter.measure		82%		71%	55	145	75	390	4	42	0	4
models.measure.note.notations.technical		11%		0%	48	57	95	106	44	53	8	9
models.measure.note		44%		50%	38	71	70	123	37	69	5	11
models.measure.note.notations		13%		0%	29	36	59	67	25	32	3	4
converter.measure_line		69%		50%	15	31	23	73	2	13	0	4
models.measure.barline		0%		n/a	21	21	30	30	21	21	3	3
models.measure.attributes		61%		n/a	14	36	28	66	14	36	1	6
models		61%		n/a	14	33	27	62	14	33	1	5
models.measure.direction		0%		n/a	15	15	21	21	15	15	3	3
models.part_list		65%		n/a	9	24	19	51	9	24	1	5
models.measure		55%		n/a	7	18	14	28	7	18	1	2
custom_exceptions		0%		n/a	4	4	8	8	4	4	4	4
Total	7,722 of 19,375	60%	715 of 1,443	50%	915	1,585	1,830	4,193	406	827	51	128

At first glance, the total coverage is 60%. Although there is 40% missing, this should be enough testing as a majority of what was not tested was from the starter project, meaning it should be assumed that it already works as intended. As such, it was decided not to test those cases. Furthermore, as described earlier, a majority of the tests are GUI tests, which might not reflect everything in the coverage report.

A majority of the code created this term can be found under the draw.score package, and as such, was the package that was tested the most.

draw.score

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
DrawGuitarNotes		83%		70%	8	26	19	123	1	9	0	1
GuitarStaff		62%		75%	1	4	12	35	0	2	0	1
DrumNotesList		94%		68%	14	30	11	85	0	8	0	1
DrawDrumNotes		98%		82%	8	33	7	156	0	10	0	1
GuitarNotesList		99%		85%	3	17	0	62	0	7	0	1
DrumStaff		100%		100%	0	3	0	6	0	2	0	1
Total	243 of 2,948	91%	36 of 150	76%	34	113	49	467	1	38	0	6

As can be seen, 91% of the package has been covered, with only minor conditionals being missed. 91% is a very high number and considering all the tests have passed, indicates that the project is well tested and can be reliably run with no error.