Group 5

TAB2XML Testing Document

April 2022

EECS 2311

Duaa Ali, Michelle Saltoun, Mohammad Paurobally, Vincent Mai, and Joshua Hanif

Table of Contents

1. Introduction

This file documents the testing done on the TAB2XML application through the JUnit framework.

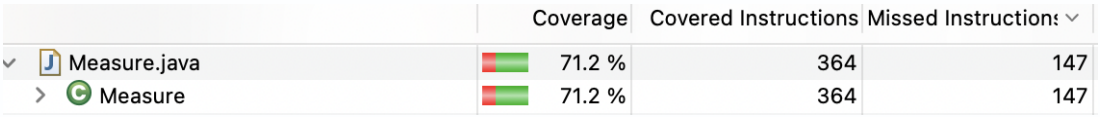
2. JUnit Testing

2.1 Parser

This parses the MusicXML string in a format where each element is easily retrievable and able to be used by other classes.
For now, we will be testing the following classes in the Parser package.
For all of the test classes below, two parsers were made; one for wikiGuitarTab and one for wikiDrumTab, in order to test the real, expected input against our cases. From here on, they will be interchangeably referred to as “input” or “parsers”.

2.1.1 Measure

testMeasureNumber(): Tests that the integer returned matches the current measure in the provided input.
Since each input had two measures, in total 4 cases were checked; two measures per parser.
The cases implemented asserted that each measure was being correctly parsed by comparing the expected measure number against the actual measure number returned by getMeasureNumber() when called on each of the two parsers.
For example, the expected value of the second measure in the wikiGuitarTab is 2 and so if wikiGuitarTab.getMeasures.get(1).getMeasureNumber() returned 2, then the test passed.
testGetNumNotes(): Tests that the integer returned matches the expected number of notes from the provided input.
For this test, all the notes in each measure were asserted independently, and thus, there were 4 test cases, two per parser.
The expected number of notes were manually counted from the input and were then compared with the actual integer returned from the getNumNotes() method when called on the parsers.
For example, in the wikiGuitarTab, measure one had 8 notes and so this number was compared against the value returned from wikiGuitarTab.getNumNotes(). If both the integers matched, then the test passed and the method parsed correctly.
These test cases were sufficient because as can be seen in the screenshot below, the tests covered the majority of the methods in the Measure class.

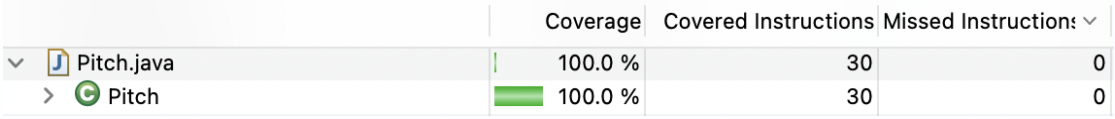


	Coverage	Covered Instructions	Missed Instructions
Measure.java	71.2 %	364	147
Measure	71.2 %	364	147

2.1.2 Pitch

Note: this test is primarily used to test the values for guitar tablature, since drum tablature uses the unpitched class to retrieve the step and octave values.
testStep(): Tests if all the expected step values match with the values the parser retrieved from the musicxml, using getStep().
This made sure that the parser was consistently reading the correct note, and to check if the parser was checking the correct step value for that specific note in the tablature.

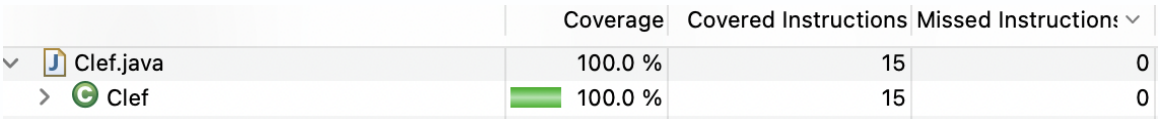
testAlter(): Tests if all of the alter values are correctly read from the musicxml using getAlter(), and checks if it matches the corresponding expected values.
This test was crucial because in some scenarios, there would be no alter value present for a specific note, in which we would expect the value returned to be zero if there was none present. This was also important because the alter value signifies if the note was supposed to include an accidental, such as a sharp or flat, which can greatly change how the note sounds.
testOctave(): Tests if all of the octave values are correctly read from the musicxml formatted text file, and matches with the values inside the musicxml. This test checks if the octave values matched the expected values from the musicxml, and checks if it is reading the octave value from the correct note in the tablature.
The code coverage for this class can be seen below. As is proven by the metrics, the test cases were sufficient.



	Coverage	Covered Instructions	Missed Instructions
Pitch.java	100.0 %	30	0
Pitch	100.0 %	30	0

2.1.3 Clef

testGetSign(): Tests if the sign of the clef returned matches the expected sign of the provided instrument.
For this test, two different test cases were tested, depending on the instrument and the measure. For the Guitar, for measure one, the expected value, “TAB”, was compared against the sign returned by getSign().
For the Drum, for measure one, the expected value, “percussion”, was compared against the sign returned by getSign().
For both instruments, the sign of the clef returned by getSign() for measure two was compared against “null”, since the second measure shouldn’t have a clef.
testGetLineValues(): Tests if the line value of the clef matches the actual placement of clef in the input.
For this test, in terms of measure one, the actual line value of the clef was manually retrieved from the input file and was then compared against the value returned by the getLineValue method. For measure two, since there was no clef, the returned value was compared against 0.
These test methods were sufficient because as can be seen below, the code coverage was 100%. Additionally, there are only two kinds of clefs; TAB and percussion, and since we tested for both of them on two different instruments, the testing was deemed sufficient.

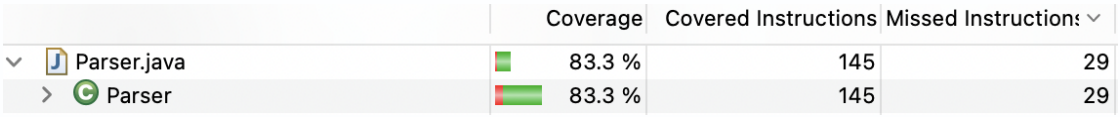


	Coverage	Covered Instructions	Missed Instructions
Clef.java	100.0 %	15	0
Clef	100.0 %	15	0

2.1.4 Parser

testGetNumMeasures(): Tests that the number of measures returned by the GetNumMeasures method matches the actual number of measures in the provided input.
To test this method, the expected number of measures were manually retrieved and compared against the integer returned by the method under test.
For example, for the Guitar tab, the number of measures in the file was 2 and so when tested with our method, 2 was also the integer returned. Since the numbers matched, the test was deemed successful and the method correct.

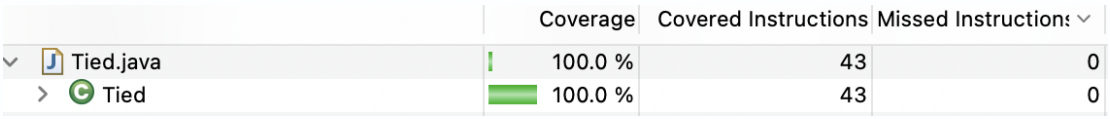
testGetInstrument(): Tests that the getInstrument method correctly identifies the instrument given in the input.
To test this method, the expected value was once again manually retrieved and then compared against the string returned by the method. Since the assertion returned true and they were indeed the same string, the method was deemed correct.
testGetArtist(): Tests that the getArtist method correctly retrieves the name of the Artist provided in the input.
For the two parsers that we've used thus far as input, as is obvious, there is no Artist and hence, the method's return value was compared against "null".
testGetTitle(): Tests that the getTitle method correctly retrieves the Title of the provided input.
Similar to the getArtist method, for both the parsers, there was no title element and thus, the method's return value was compared against "null".
The testing was deemed sufficient because of the following code coverage metrics for the Parser class which proved that the majority of the code was covered with our cases:



	Coverage	Covered Instructions	Missed Instructions
Parser.java	83.3 %	145	29
Parser	83.3 %	145	29

2.1.5 Tied

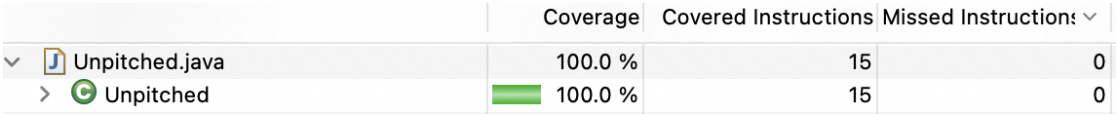
testGetters(): Tests that the values returned by the getStart(), getStop(), getCont(), and getLetRing() methods match the expected values for these elements.
For this test, since wikiGuitarTab and wikiDrumTab had no tied element, instead of using our generic input as we have thus far, a tiedTest1 parser was used instead. Although this file had four measures, for testing purposes, in order to save time and still be effective, only the first two measures were considered as input.
To test these methods, the expected value was manually retrieved from the file and then compared with the actual value returned by each method. This was repeated twice, once for each measure.
testSetters(): Tests that the values returned by the getStart(), getStop(), getCont(), and getLetRing() methods match the expected values once the values are updated by setStart(), setStop(), setCont(), and setLetRing(), respectively.
Initially, the values of all the methods were false (this was verified by the test above).
Then, to test the setter methods, the values of all four methods were updated to equal true by their respective setters before the assertion statements. Then, using the getter methods, the values were checked against the new values.
These test methods were deemed sufficient because as can be seen below, the code coverage was 100%



	Coverage	Covered Instructions	Missed Instructions
Tied.java	100.0 %	43	0
Tied	100.0 %	43	0

2.1.6 Unpitched

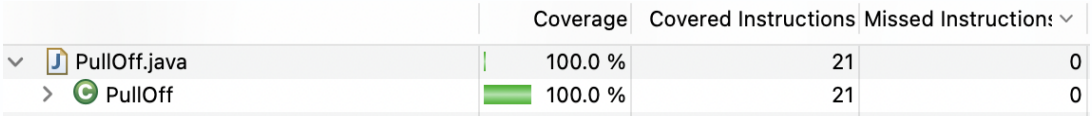
testGetStep(): Tests that the step returned by the getStep method matches the expected step of a note.
To test this method, all step variations were considered through looping if-statements. The step values of all the notes in the wikiDrumTab were manually retrieved and compared against the returned value provided by the getStep method. This was done for both measures one and two.
testOctave(): Tests that the octave returned by the getOctave method matches the expected octave of a note.
Similar to testGetStep(), to test this method, all octave variations were considered through looping if-statements for both measures in the file. Note that the measures were tested separately in order to keep the testing code easy to read and comprehend.
These test cases were deemed sufficient because as can be seen below, the code coverage was 100%.



	Coverage	Covered Instructions	Missed Instructions
Unpitched.java	100.0 %	15	0
Unpitched	100.0 %	15	0

2.1.7 PullOff

For the following three tests, a capricho parser was used, instead of the wikiGuitarTab and wikiDrumTab parsers since those files had no pull-off elements. Also note that since the capricho file was extensive, only the first two measures were considered for testing.
testGetNumber(): Tests that the number returned by the getNumber method matches the expected pull-off number for the given note.
To test this method, an ArrayList was created which stored all the notes from the first measure. Then, while looping through this list, the pull-off number returned by the method was asserted against the expected pull-off number retrieved through manual means. Since the first measure had no pull-off elements for its notes, the returned value was compared with 0.
For measure two, a similar process was followed, but since this measure actually had notes with pull-off elements, each returned value was individually asserted against the expected pull-off number of each note.
testGetType(): Tests that the type returned by the getType method matches the expected pull-off type for the given note.
To test this method, a similar process to the one above was followed. All measure one notes were asserted against “null” for pull-off types and all notes from the second measure were individually matched against the expected string, which was either “start”, “stop”, or “null” depending on the note under test.
testGetValue(): Tests that the value returned by the getValue method matches the expected pull-off value for the given note.
As expected, the same procedure was followed for this test as well. All measure one notes were asserted against “null” for pull-off values and all notes from the second measure were individually matched against the expected string, which was either “null” or “P” depending on whether the note under test had a pull-off value or not.
These test cases were deemed sufficient because as can be seen below, the code coverage was 100%.



	Coverage	Covered Instructions	Missed Instructions
PullOff.java	100.0 %	21	0
PullOff	100.0 %	21	0

2.1.8 Slur

Similar to the PullOffTest class, for the following three tests, a capricho parser was used instead of the wikiGuitarTab and wikiDrumTab parsers since those files had no slur elements. Also note that since the capricho file was extensive, only the first two measures were considered for testing.

testGetNumber(): Tests that the number returned by the getNumber method matches the expected slur number for the given note.

To test this method, an ArrayList was created which stored all the notes from the first measure. Then, while looping through this list, the slur number returned by the method was asserted against the expected slur number retrieved through manual means. Since the first measure had no slur elements for its notes, the returned value was compared with 0.

For measure two, a similar process was followed, but since this measure actually had notes with slur elements, each returned value was individually asserted against the expected slur number of each note.

testGetType(): Tests that the type returned by the getType method matches the expected slur type for the given note.

To test this method, a similar process to the one above was followed. All measure one notes were asserted against "null" for slur types and all notes from the second measure were individually matched against the expected string, which was either "start", "stop", or "null" depending on the note under test.

testGetPlacement(): Tests that the value returned by the getPlacement method matches the expected slur placement for the given note.

As expected, the same procedure was followed for this test as well. All measure one notes were asserted against "null" for slur placements and all notes from the second measure were individually matched against the expected string, which was either "null" or "above" depending on whether the note under test had a slur or not.

These test cases were deemed sufficient because as can be seen below, the code coverage was 100%.

	Coverage	Covered Instructions	Missed Instructions
Slur.java	100.0 %	21	0
Slur	100.0 %	21	0

2.1.9 Attributes

testDrumTab(): Tests whether or not the actual value of the clef, fifths and divisions of the attributes class matches its expected value.

testGuitarTab(): Tests whether or not the actual value of the clef, fifths and divisions of the attributes class matches its expected value.

Both sets of tests verifies the value of each accessor method of the Attributes.java class in two ways.

1. To directly retrieve the value of the divisions, fifths, clef and compare them with their expected values.
2. To test for the occurrence of an IndexOutOfBoundsException when accessing the clef at an invalid index.

Attributes.java	94.2 %	98	6	104
Attributes	94.2 %	98	6	104

2.1.10 Note

testGetDuration(): Tests that the duration returned by the GetDuration method matches the actual duration of the notes in the provided input.
To test this method, the expected duration was manually retrieved and compared against the integer returned by the method under test for both guitar and drum parsers.
For example, for the Guitar tab, every note in the first measure had a duration of 8 and so when tested with our method, 8 was also the integer returned. Since the numbers matched, the test was deemed successful and the method correct.
testGetVoice(): Tests that the voice returned by the GetVoice method matches the actual voice of the notes in the provided input.
To test this method, the expected voice was manually retrieved and compared against the integer returned by the method under test for both guitar and drum parsers.
For example, for the Guitar tab, every note in the first measure had a voice of 1 and so when tested with our method, 1 was also the integer returned. Since the numbers matched, the test was deemed successful and the method correct.
testGetType(): Tests that the type returned by the getType method matches the expected type for the given note.
To test this method, a for-loop was run and the type of each note was compared against the expected type of the note.
For example, for the Guitar tab, every note in the first measure was an eight note, and so its type, "I", was asserted against the char returned by the method under test. Since they both matched, the test was deemed successful and the method correct.
The same for-loop was also run for drum tabs!
6 is the bottom string
testGetString(): Tests whether the line number (referred to as string) returned by the getString method matches the actual line that the note is supposed to be placed on.
To test this method, a for-loop was created which iterated through all the notes from the first measure. Then, while looping through this list, the line number returned by the method was asserted against the expected line number retrieved through manual means.
Note that the number "1" referred to the first line of the staff and the number "6" referred to the last line of the guitar staff.
testGetFret(): Tests that the fret returned by the GetFret method matches the actual fret of the notes in the provided input.
To test this method, the expected fret was manually retrieved and compared against the integer returned by the method under test.
For example, for the Guitar tab, each note in the first measure had a fret value of either 0, 1, 2, 3, and 7. Thus, the fret values returned by the method under test were compared against their respective expected values. Since the numbers matched, the test was deemed successful and the method correct.

testIsChord_SetChord(): Tests that the boolean returned by the isChord method accurately checks if the notes are a chord.

To test this method, a for-loop was created which iterated through all the notes from the first measure. Then, while looping through this list, the notes were compared to see if they formed a chord. Depending on this, the value returned by the method was asserted against the expected boolean retrieved through manual means.

testGetInstrumentID(): Tests that the getInstrument method correctly identifies the instrument given in the input.

To test this method, the string returned by the method was compared against a list of different types of drums in order to determine if the method was retrieving the correct one. Since the assertion returned true and they were indeed the same instrument, the method was deemed correct.

testGetNotehead(): Tests that the notehead returned by the GetNotehead method matches the actual notehead in the provided input.

To test this method, the expected notehead value was manually retrieved and compared against the integer returned by the method under test for the drum parser.

testGetBendAlter(): Tests that the bend alter returned by the GetBendAlter method matches the actual bend alter in the provided input.

To test this method, instead of the usual two parsers, a bendTest1 parser was made. Then, the value returned by the method under test was asserted against the expected value, which as always, was retrieved through manual parsing.

testGetNumDots(): Tests that the number of dots returned by the GetNumDots method matches the actual number of dots for a specific note in the provided input.

To test this method, instead of the usual two parsers, a parabola parser was made. Then, the method was run using the parser and the value returned by the method under test for each note that had dots was asserted against the expected value, which as always, was retrieved through manual parsing.

testIsGraceNote_SetGraceNote(): Tests that the boolean returned by the isGraceNote method accurately checks if some set of notes have a grace.

To test this method, instead of the usual two parsers, a capricho parser and graceTest1 parser was made. Then, the method was run using the two parsers separately and the boolean returned by the method under test for each set of notes that had a grace was asserted against the expected set of notes, which as always, was retrieved through manual parsing.

testIsRest_SetRest(): Tests that the boolean returned by the isRest method accurately checks if some note is a rest.

To test this method, instead of the usual two parsers, a parabola parser was made. Then, the method was run using the parser and the boolean returned by the method under test for each note that was a grace was asserted against the expected notes, which as always, was retrieved through manual parsing.

testIsNatural_SetNatural(): Tests that the boolean returned by the isNatural method accurately checks if a note is a natural.

To test this method, instead of the usual two parsers, a capricho parser was made. Then, the method was run using the parser and the boolean returned by the method under test for each note that was a natural was asserted against the expected notes, which as always, was retrieved through manual parsing.

testIsArtificial_SetArtificial(): Tests that the boolean returned by the isArtifical method accurately checks if some note is artificial.

To test this method, instead of the usual two parsers, a capricho parser was made. Then, the method was run using the parser and the boolean returned by the method under test for each

note that was artificial was asserted against the expected notes, which as always, was retrieved through manual parsing.

testGetNoteheadParentheses_SetNoteheadParentheses(): Tests that the boolean returned by the getNoteheadParentheses method accurately checks if some note has parentheses.

To test this method, instead of the usual two parsers, a parabola parser was made. Then, the method was run using the parser and the boolean returned by the method under test for each note that had a parentheses was asserted against the expected notes with parentheses, which were retrieved through manual means.

These test cases were deemed sufficient because as can be seen below, the code coverage was 100% for the Note class.

	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Note.java	82.3 %	232	50	282
Note	82.3 %	232	50	282

2.1.11 Barline

Note that for all of the following cases, instead of using the drum and guitar tabs as we have thus far, we made parsers for the repeat and money files, since they are the only classes with repeats in them!

testGetLocation(): Tests whether the location (either left or right) returned by the getLocation method matches the expected location of the barlines.

To test this method, the expected location was visually retrieved for a repeat measure using Musescore and compared against the char returned by the method under test.

testRepeatDirection(): Tests whether the direction (forward or backward) returned by the getRepeatDirection method matches the expected direction.

To test this method, the expected direction was manually retrieved for a repeat measure and compared against the char returned by the method under test.

testGetBarStyle(): Tests that the bar style (either “heavy-light” or “light-heavy”) returned by the GetBarStyle method matches the actual style of the measures in the provided input that have repeats.

To test this method, the expected bar style was manually retrieved for a repeat measure and compared against the string returned by the method under test.

testGetRepeatTimes(): Tests that the repeat number returned by the GetRepeatTimes method matches the actual repeat number of the measures in the provided input.

To test this method, the expected repeat number was manually retrieved for a certain measure and compared against the integer returned by the method under test.

For example, for the tab from the repeat file, the second measure had a repeat times 7, thus, this number was asserted against the number returned by the method for this same measure. Since the numbers matched, the test was deemed successful and the method correct.

These test cases were deemed sufficient because as can be seen below, the code coverage was 100% for the Note class.

	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Barline.java	90.3 %	56	6	62
Barline	90.3 %	56	6	62

2.1.12 Direction

Note that for all of the following cases, instead of using the drum and guitar tabs as we have thus far, we made parsers for the repeat and money files, since they are the only classes with repeats in them!

testGetPlacement(): Tests whether the placement (either null or above) returned by the getPlacement method matches the expected placement of the repeat text.

testGetX(): Tests whether the x value of the repeat text returned by the getX method matches the expected x value of the repeat text.

To test this method, the expected x value was manually decided for a certain measure and compared against the integer returned by the method under test.

testGetY(): Tests whether the y value of the repeat text returned by the getY method matches the expected y value of the repeat text.

To test this method, the expected y value was manually decided for a certain measure and compared against the integer returned by the method under test.

testGetWords(): Tests whether the string of the repeat text returned by the getWords method matches the expected string value of the repeat text.

To test this method, the expected word value (for example, "x7") was manually retrieved for a certain repeat measure and asserted against the string returned by the method under test.

These test cases were deemed sufficient because as can be seen below, the code coverage was 100% for the Note class.

	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Direction.java	91.3 %	42	4	46
Direction	91.3 %	42	4	46

2.1.13 TimeModifications

To test the following methods, a capricho parser was made and used!

testGetActual(): Tests whether the actual notes returned by the GetActual method match the expected actual notes in the provided input.

To test this method, the expected actual notes were manually retrieved and compared against the notes returned by the method under test for the capricho tablature.

For example, the 11th measure of the tab had actual notes in it and so the getter was run on this measure in order to see if it would accurately label them as actual notes. There were two values that were asserted, -1 (not actual note) and 3 (value of the actual notes), against the values returned by the method for each note in that specific measure.

testGetNormal(): Tests whether the normal notes returned by the GetNormal method match the expected normal notes in the provided input.

To test this method, the expected normal notes were manually retrieved and compared against the notes returned by the method under test for the capricho tablature.

For example, the 11th measure of the tab had normal notes in it and so the getter was run on this measure in order to see if it would accurately label them as normal notes. There were two values that were asserted, -1 (not normal note) and 2 (value of the normal notes), against the values returned by the method for each note in that specific measure.

These test cases were deemed sufficient because as can be seen below, the code coverage was 100% for the Note class.

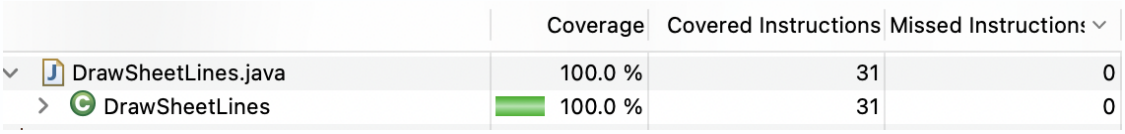
	Coverage	Covered Instructions	Missed Instructions	Total Instructions
TimeModification.java	100.0 %	15	0	15
TimeModification	100.0 %	15	0	15

2.2 GUI

This package is quite difficult to test due to its nature and hence the only class that really needed testing is the DrawSheetLines class.

2.2.1 DrawSheetLines

testGetLine(): Tests that the coordinates of the line provided are the same coordinates returned by the getter method.
To test this method, we first created a DrawSheetLines object, which was essentially just a line. We then retrieved the individual x and y coordinates through the getter method under test and asserted them with the expected values.
Although 4 assertions were written, this was all just one test case since they all belonged to the same object.
This one test case was sufficient for the entire method because if the getter works for one line, then consequently, it'll work for any and all lines since the only difference between any two lines is their x and y coordinates, which we have already tested.
testSetLine(): Tests that the coordinates of the line are successfully updated through the setter method.
To test this method, we first created a DrawSheetLines object (named sheetLine) and initialized it by giving it some random values. Then, we created a new line with new values (named line) through the Line class imported from JavaFx.scene.shape. Now, to test the setter method, we updated the values of "sheetLine" to the values of "line" by calling the setLine method on "sheetLine". These values were then asserted with the expected values and if they passed, the testing proved that the setter method worked correctly.
Once again, testing on one line was sufficient because if it works for one line, consequently, it'll work for all other lines as well.



	Coverage	Covered Instructions	Missed Instructions
DrawSheetLines.java	100.0 %	31	0
DrawSheetLines	100.0 %	31	0