Tab2Pdf Testing Documentation

EECS 2311
V. Tzerpos
Wednesday, February 25, 2015

Group 1
Atto, Brody
Cirillo, Marco
Patel, Deep
Ragavendran, Varsha
Sitiugin, Glib
Sitkovets, Anton

Testing Methodology

Since Tab2Pdf is built using the agile software development method, we opt to incorporate Test Driven Development into our sprints. When a group member commits code to the repository (via a pull request), the corresponding test cases are also included. This can be seen as a “bottom up” approach to testing, as the foundations of the software are tested as they are built.

Locally, when a member is working on a new feature, stub methods for the feature are made first, followed by test cases. These test cases are meant to fail, since no code was written yet. As the developer builds out the code in the former stub methods, tests begin to pass, and the software is assured to work correctly.

In our tests, we supply a minimum of 5 valid inputs and 5 invalid inputs to each test case. Examples of valid input were taken straight from the sample files on the course website. Invalid input was created by malforming the valid input, such as adding characters that were not valid (`%`), making spelling mistakes (`ATHOR=` instead of `AUTHOR=`).

This document will detail the specific methods in our JUnit test files, their importance to the overall operation of the code, and the metrics of our testing.

Testing Sufficiency

We believe fully testing the models and parsers are sufficient because they represent the business logic of this program. These classes are responsible for the bulk of what Tab2PDF does, and as such, should be thoroughly tested.

The GUI could have been tested with Java's `Robot` class, but we opted for manual testing, since the GUI is rather simplistic (and Sun thoroughly tests Swing code, so we know we have a solid foundation).

`Util` classes are also untested, as they are simple functions.

The PDF classes `PdfHelper` and `PdfCreator` are untested because there is difficulty in comparing PDF documents. This is because two documents can be rendered differently in different PDF viewers, so it is tough to gauge which viewer is the truly correct (ie: the *oracle*).

Test Classes

IParser Test Classes

There are two methods we test in all parsers (classes that implement `IParser`). The first of which is `canParse(String token)` which returns true if the parser can parse the given token into an `ITabNotation` object. The `ITabNotation` object represents a single piece of the tab, such as a note, slide, dash or hammer-on. The second method we test is the `parse(String token)` method, which actually parses the token and returns an `ITabNotation` object, throwing a `ParseException` if the token cannot be parsed. If `canParse()` returns `true`, it is expected that `parse()` will not throw a `ParseException`.

DashParserTest.java

This class tests the programs ability to parse a dash (`-`) in tab notation. A dash signifies a space between notes. The more dashes there are in a row, the longer the space should be. For valid lines, we ensure the line begins with at least one dash. For invalid lines, we put characters that are *not* dashes at the beginning of the line, such as `|`, `1`, `||`, `_`, and `%`.

DoubleBarParserTest.java

This class tests the programs ability to parse a repeat bar, which has a few forms:

Standard Repeat Bar

This should render as a regular repeat bar

```
||- ... -||
||- ... -||
||* ... *||
||* ... *||
||- ... -||
||- ... -||
```

Repeat Bar with a Specified Number of Repeats

This should render as a repeat bar with “Repeat 3 Times” at the top of it

```
||- ... -|3
||- ... -||
||* ... *||
||* ... *||
||- ... -||
||- ... -||
```

A repeat bar signifies a bar should repeat (an optional amount of times). For valid lines, we ensure the line begins with at least one type of repeat bar. For invalid lines, we put characters that are *not* valid repeat bars at the beginning of the line, such as `-`, `1`, `**||`, `-|`, and `|`.

NoteParserTest.java

This class tests the programs ability to parse a note (`3`) in tab notation. A note signifies a musical note. Valid notes should be between 0 and 24 inclusive, as most guitars have a maximum of 24 frets. For valid lines, we ensure the line begins with at least one number. For invalid lines, we put characters that are *not* numbers at the beginning of the line, such as `|`, `-`, `||`, `_`, and `%`.

SlideParserTest.java

This class tests the programs ability to parse a slide (`3s5`) in tab notation. A slide signifies a slide between musical notes. The slide must have at least one start note or end note. A slide is considered indeterminate if it does not have both a start and end note (`s4`), and it is up to the musician to determine where to begin the slide from. Valid slides should have valid notes between 0 and 24 inclusive, as most guitars have a maximum of 24 frets. For valid lines, we ensure the line begins with at least one number or a slide (`s`). For invalid lines, we put characters that are *not* numbers at the beginning of the line, such as `|`, `-`, `||`, `12`, and `%`.

SpacingParserTest.java

This class tests the programs ability to parse a spacing attribute (`SPACING=1`) in tab notation. A spacing signifies the horizontal spacing the PDF output should follow. A higher spacing value creates more space between notes. For valid lines, we ensure the line begins with at least (case-insensitive) `SPACING=N`, where N is any integer or decimal number. For invalid lines, we put characters that are *not* numbers or misspell/shorthand `SPACING`, such as `SPACING=`, `SPACING=A`, `|--`, `0--|`, and `S=1`.

SquareNoteParserTest.java

This class tests the programs ability to parse a square note (`<2>`) in tab notation. A square note signifies a harmonic should be played. For valid lines, we ensure the line begins with at least `<N>`, where N is any integer. For invalid lines, we put characters that are *not* numbers in angle brackets, such as `|`, `-`, `1`, `23`, and `%`.

SubtitleParserTest.java

This class tests the programs ability to parse a title attribute (`SUBTITLE=Moonlight Sonata`) in tab notation. A title signifies the subtitle of the tab (typically the author). For valid lines, we ensure the line begins with at least (case-insensitive) `SUBTITLE=S`, where S is any string. For invalid lines, we put empty strings as the title or misspell/shorthand `TITLE`, such as `TITLE=`, `T=A`, `SUBTITLE=`, `Jim Matheos`, and `S=A`.

TabParserTest.java

This class tests the programs ability to parse an entire tab file (Moonlight Sonata) into tab notation. The tab file includes title, subtitle, and spacing attributes as well as the entire tab.

TitleParserTest.java

This class tests the programs ability to parse a title attribute (`TITLE=Jim Matheos`) in tab notation. A title signifies the title of the tab. For valid lines, we ensure the line begins with at least (case-insensitive) `TITLE=S`, where S is any string. For invalid lines, we put empty strings as the title or misspell/shorthand `TITLE`, such as `TITLE=`, `T=A`, `SUBTITLE=`, `Remembering Rain`, and `S=A`.

ITabNotation Test Classes

These test classes test all of our model objects (those that implement `ITabNotation`). These represent each symbol that we support in tab notation.

BarLineTest.java

This class fully tests the `BarLine` class (representing a line in a `Bar`), including mutator methods (`getters` and `setters`) and equals.

BarTest.java

This class fully tests the `Bar` class (representing a musical `Bar`), including mutator methods (`getters` and `setters`) and equals.

DoubleBarTest.java

This class fully tests the `DoubleBar` class (representing a repeating Bar), including mutator methods (`getters` and `setters`) and equals.

HammerOnTest.java

This class fully tests the `HammerOn` class (representing a musical hammer on such as `3h5`), including mutator methods (`getters` and `setters`) and equals.

PullOffTest.java

This class fully tests the `HammerOn` class (representing a musical pull off such as `5p3`), including mutator methods (`getters` and `setters`) and equals.

ScalingTest.java

This class fully tests the `Scaling` class (representing tab vertical scaling, such as `SCALING=5.0`), including mutator methods (`getters` and `setters`) and equals.

SpacingTest.java

This class fully tests the `Spacing` class (representing tab horizontal spacing, such as `SPACING=5.0`), including mutator methods (`getters` and `setters`) and equals.

SquareNoteTest.java

This class fully tests the `SquareNote` class (representing a harmonic `Note`), including mutator methods (`getters` and `setters`) and equals.

SubtitleTest.java

This class fully tests the `Subtitle` class (representing the tab's subtitle such as `SUBTITLE=Ludwig van Beethoven`), including mutator methods (`getters` and `setters`) and equals.

TabTest.java

This class fully tests the `Tab` class (representing the ASCII musical tab as a whole), including mutator methods (`getters` and `setters`) and equals.

TitleTest.java

This class fully tests the `Title` class (representing the tab's title such as `TITLE=Moonlight Sonata`), including mutator methods (`getters` and `setters`) and equals.

Coverage and Metrics

Code coverage is grouped by package. Our focus is on 100% method coverage in model and parser classes. We chose not to perform automated testing of the GUI, instead opting for manual testing.

ca.yorku.cse2311.tab2pdf.model

This table showcases the testing coverage for our model classes.

Class	Class %	Method %	Line %
Bar	100% (1/1)	100% (18/18)	92.9% (39/42)
BarLine	100% (1/1)	100% (7/7)	100% (15/15)
Dash	100% (1/1)	100% (8/8)	100% (13/13)
DoubleBar	100% (1/1)	100% (13/13)	84.6% (44/52)
HammerOn	100% (1/1)	100% (13/13)	92.9% (26/28)
Note	100% (1/1)	100% (10/10)	90.5% (19/21)
Pipe	100% (1/1)	100% (7/7)	100% (8/8)
PullOff	100% (1/1)	100% (13/13)	92.9% (26/28)
Scaling	100% (1/1)	100% (10/10)	100% (18/18)
Slide	100% (1/1)	100% (12/12)	93.9% (31/33)
Spacing	100% (1/1)	100% (10/10)	100% (18/18)
SquareNote	100% (1/1)	100% (9/9)	89.5% (17/19)
Subtitle	100% (1/1)	100% (10/10)	100% (17/17)
Tab	100% (1/1)	100% (17/17)	100% (33/33)
Title	100% (1/1)	100% (10/10)	100% (17/17)

ca.yorku.cse2311.tab2pdf.parser

This table showcases the testing coverage for our parser classes.

Class	Class %	Method %	Line %
AbstractParser	100% (1/1)	100% (2/2)	100% (2/2)
DashParser	100% (1/1)	100% (4/4)	100% (7/7)

Class	Class %	Method %	Line %
DoubleBarParser	100% (1/ 1)	100% (4/ 4)	92.9% (26/ 28)
HammerOnParser	100% (1/ 1)	100% (4/ 4)	87.5% (7/ 8)
NoteParser	100% (1/ 1)	100% (4/ 4)	100% (7/ 7)
PipeParser	100% (1/ 1)	100% (4/ 4)	100% (7/ 7)
PullOffParser	100% (1/ 1)	100% (4/ 4)	87.5% (7/ 8)
SlideParser	100% (1/ 1)	100% (4/ 4)	100% (9/ 9)
SpacingParser	100% (1/ 1)	100% (4/ 4)	85.7% (6/ 7)
SquareNoteParser	100% (1/ 1)	100% (4/ 4)	100% (7/ 7)
SubtitleParser	100% (1/ 1)	100% (4/ 4)	85.7% (6/ 7)
TabParser	100% (1/ 1)	100% (11/ 11)	76.8% (96/ 125)
TitleParser	100% (1/ 1)	100% (4/ 4)	85.7% (6/ 7)

Tab2Pdf

This table showcases the testing coverage for our entire program, grouped by package.

Class	Class %	Method %	Line %
ca.yorku.cse2311.tab2pdf	0% (0/ 3)	0% (0/ 16)	0% (0/ 57)
ca.yorku.cse2311.tab2pdf.model	100% (15/ 15)	100% (167/ 167)	94.2% (341/ 362)
ca.yorku.cse2311.tab2pdf.parser	100% (13/ 13)	100% (57/ 57)	84.3% (193/ 229)
ca.yorku.cse2311.tab2pdf.parser.exception	33.3% (1/ 3)	33.3% (1/ 3)	33.3% (2/ 6)
ca.yorku.cse2311.tab2pdf.pdf	50% (1/ 2)	47.2% (17/ 36)	45.3% (131/ 289)

Class	Class %	Method %	Line %
ca.yorku.cse2311.tab2pdf.ui	0% (0/ 2)	0% (0/ 26)	0% (0/ 86)
ca.yorku.cse2311.tab2pdf.ui.component	0% (0/ 6)	0% (0/ 55)	0% (0/ 244)
ca.yorku.cse2311.tab2pdf.ui.listener	0% (0/ 12)	0% (0/ 32)	0% (0/ 129)
ca.yorku.cse2311.tab2pdf.util	0% (0/ 4)	0% (0/ 14)	0% (0/ 45)