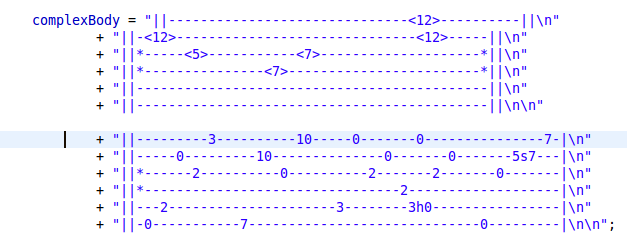
Testing

The testing procedures were largely developed in tandem with the code since we were encouraged to use test driven development. The team was not experienced with this kind of development, but it proved to be very useful upon refactoring. We were able to make major changes to the design with confidence that it would not break our code. The unit tests were helpful only to a certain point.

System testing presented a challange because other than checking the output “by eye” there was no other way we could cognize that the output corresponded correctly with the acceptance tests. Rather than testing the whole system when only small portions of the code needed to be tested for proper output. Snippits of the acceptance tests input were embedded into the testing framework and setup before each testing method call. This saved considerable time, since opening a file took significantly longer to do than simplying pressing the JUnit run button.

Illustration 1: Example of a test case embedded in a test class

Different levels of complexity were used for testing various methods. For instance the first test input called “simpleBody” contains only single vertical bars, no repeats and only one stave. The others contain multiple staves, staves of varying lengths and repeat numbers embedded within the bars. This strategy enabled us to gradually refine our code to test against increasingly complex input. Similar to what in done in formal methods, but not near as rigourious.

There are parts of the code which need more thorough testing however. A large amount of testing was done only to ensure that we could make changes to the system without breaking it. There should be at least two test cases per method, one which should succeed and one which should fail. But most tests only test for the success of a method. Certain cases were methods were trivial to implement or creating a test case got in the way of the rythym of development of a particular part of the system, a test was not made. A important aspect of testing was perhaps not given attention. As bugs were found in the system and fixed. New test cases were usually not implemented. The problem could return if that particular piece of code was changed again and the bug may re-appear with no test to catch it. As the project is further refined and more features are added, this is a point we will not ignore. As maintenence is extremely important to the software development life cycle.

Throwing and catching exceptions was not acknowledge at this point in the project. Our goal was to get the system working first, then consider special circumstances of how to handle faults. We plan to also make test cases for exceptions, so we canbe sure that they are handled correctly.

Partition testing is what we should be doing, however in many of the system tests this was not possible because we had only the deisred result of the acceptance tests to use as an oracle. Parition testing could be done on the algirthims for re-organizing the staves and sizing and will most likely be used more as the system is improved. There are important boundries or limitiations which the customer must be aware of. Things such as the minimum and maximum lengths of measures, the variety of musical symbols which can be drawn. What kind of fault tolerance is built into the system? These issues still need to be addressed and are more directly applicable to partition tests.