JythonMusic Testing Framework

Joey Baldwin, Eric Hofesmann, Marge Marshall

Abstract

JythonMusic is an open source environment for creative music making and programming, written in the Python programming language. It uses Jython, a separate software, to compile Python into Java bytecode, allowing for Java dependencies. We developed an automated testing framework built by modifying Python's unit testing library. This framework allows method testing through the use of simple testCase.txt files that can be created using a specified format. More intensive class testing can also be accomplished through the use of drivers. We created 26 test cases to exercise this framework testing the music.py, gui.py, and timer.py modules.

JythonMusic

JythonMusic is an open source environment for creative music making and programming. It was developed and is maintained at the College of Charleston. The principle investigator for the software is Dr. Bill Manaris. JythonMusic has an interesting architecture because most of it is written in the Python programming language, but it is compiled into Java bytecode. This allows programmers to use the relatively simple syntax of Python with Java dependencies. Jython, a separate open source software product, is responsible for performing the compilation.

Example Test Case

<Test Case ID>

<Description of Function's Purpose>

<Module Name> <Class Name If Applicable>

<Function Name>

<Inputs>

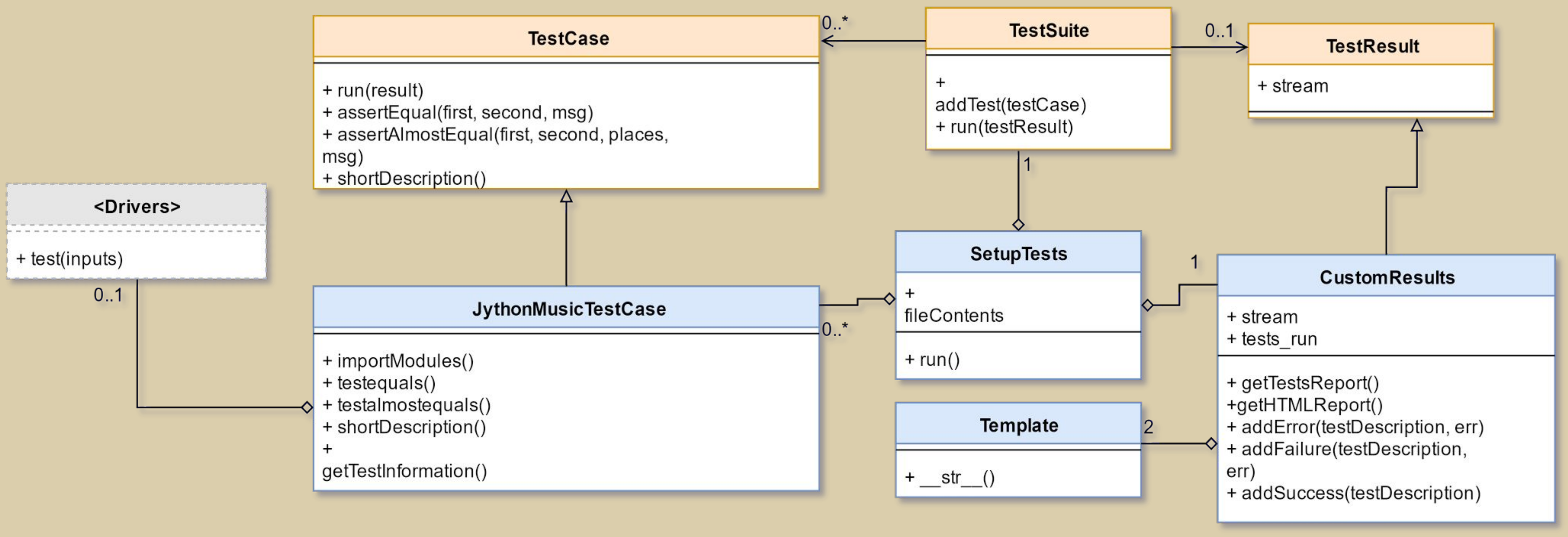
<Test Type of Expected Output>

<Expected Output>

Test Case Input Format

The test case id is designated with any number of digits, and starts with the first test case at 00. The description is an English explanation of what the function does. The module name is the module or file in which the function is found. The class name is the name of the class the function is in, if it is in a class. The function name is the name of the function being tested. The input is the current value or collection of parameters being passed to the function. The test type dictates what type of assertion will be used to determine if the actual output passes or not. This is given in the form of the name of the test functions, such as testexception or testequals. The expected output is what the function should return, given the input parameters.

Test Framework Model



The framework model utilizes classes from Python's **unittest** module (depicted in orange) in conjunction with custom classes (depicted in blue). The data flow begins with **SetupTests** which is instantiated with data parsed from the test case text files. The run function of **SetupTests** creates an instance of **TestSuite** and then proceeds to add instances of **JythonMusicTestCase** to the suite. Once the suite is populated with all of the test cases, **SetupTests** calls **TestSuite**'s run function with an instance of **CustomResults**.**TestSuite**'s run function runs each of the test cases with **CustomResults** which contains the logic for creating an HTML doc for the results

Number	Description	Module	Function	Type	Inputs	Expected	Actual	
00	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[69]	440.0	440.0	success
01	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[48]	130.81	130.81278265	success
02	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[72]	523.25	523.251130601	success
03	calculate the frequency value of a given midi pitch	music	noteToFreq	testalmostequals	[1]	8.66	8.66195721803	success

Output Format

The output of the testing framework is in the form of a table, where each test case is identified and described with its parameters, the expected outcomes, and the actual outcomes. This is valuable for the user to be able to verify that the testing framework works as expected. Successes, failures and errors are color coded so that the user can quickly scan the results.

Conclusion

The current version of this testing framework contains enormous potential. The framework utilized the most generalized testing capabilities that time allowed. That being said, there still exist many features that could imporove this design. Future work would include creating the capabilities for teseting setter methods within classes and generalizing the class testing to remove the need for drivers. Implementation of this testing software remains a destinct possibility since JythonMusics creator, Dr. Bill Manaris, works in this same institution.

References: https://jythonmusic.me/, https://github.com/ehofesmann/Cygnus-Inter-Anates