CS308 Building Software Systems

JUnit Testing Strategy

[MW-Wed-Team-4]||[20/2/18]

Identifying JUnit Test Cases

In the model package of the Gizmo-ball system, we intend to implement a fair amount of JUnit test cases to test the development of the system thoroughly. We may also use a test suite in Intellij to run multiple unit tests at once. We also intend to use the testing coverage tool to test line coverage and aim to have a high percentage of code coverage in the automated tests.

In our implementation we intend to override the equals and hash code methods and test these vigorously using the JUnit tests, testing equality of various objects. We will test for each relevant functional requirement, and test that these meet the validation of the initial system specification. This will ensure the development does what it is supposed to do and checks errors in the system.

Implementation of unit test cases to test equality, symmetry, transitivity and reflexivity. Testing objects against one another to test for both equality and equivalence. The Java API provides multiple test case methods which we will utilize such as assert equals, assert false, assert not null, and assert true.

As part of identifying what the main use cases will be to unit test the model classes we have to identify the main functional requirements and determine which parts of the program are most critical and such tests that will be implemented to gauge the correctness of the model implementation.

Due to these requirements we would like to test functionality such as:

- load game data, loading the previously saved file works as expected
- the gizmo's display in the interface as expected
- the absorber shoots the ball up the game interface and will start to lose velocity based on gravity and possibly friction, which will be tested using JUnit methods
- flipper movement: freely rotates 90 degrees upon key press, and will stay at that angle until the key release event is triggered
- ball movement object will be tested in both collision details and the velocity
- build mode functions such as add gizmo, add flipper, setting the friction and gravity values will be tested using various JUnit methods
- aim for at least 70% code coverage in specific model methods in the system

Our Approach to Junit Testing

Using the test-driven development approach, we intend to code a little, test a little and continue this process throughout the development phase. Running the tests frequently and often will also identify any new bugs found within the system which can then be changed.

Ideally when coding the main implementation of the system, we would like to write the test cases first and then code the system to meet the requirements of the test cases. Forcing us to think about the main requirements and designing the system to meet those requirements.

We will also run our test suites every so often throughout the implementation, as code that may have previously passed the unit test may at a later time fail, due to modifications in the code.

Our testing will gather the expected output versus the actual output, and we plan to analyze the results and alter any system failures that may be found, resulting in a more effective and greater performing program, ready for deployment.