

CS 1632 – Deliverable 5

Performance Testing 

*This very super cool project can be found at*

https://github.com/pahk0/1632-Project-5

Michael Fieseher

It’s pronounced “Fisher”

Summary

To find the problem areas in the program I initially thought about just looking at the code itself, but quite frankly there was so much of it I decided to save myself the trouble and just use VisualVM’s sampler tool to find which methods were causing problems. Turns out most of the computations in this program are done in MainPanel. The top one was a method called **convertToInt()**. Just based on the name, I had a feeling that this method was beyond inefficient and would be fairly easy to refactor. I was right. In fact, it’s not even necessary, since the thing it’s converting to an int is already an int. What I didn’t realize until I wrote the tests, however, is that the method does not in fact work for negative numbers. I considered this an error and fixed said error with my faster code, which I am noting here. The convertToInt() method now should work for all integers.

The second method I fixed was **runContinuous()**. I checked out this method’s big loop and tried way too hard to follow what it did. I realized a couple minutes later that it didn’t matter what the loop in the middle did, since the value just got changed right back to what it was at the beginning. I removed the try/catch and the for-loop in the middle of the method, all of which slowed it down without accomplishing any meaningful calculation.

I had a bit more trouble finding a third major issue. displayIteration() was the next-most CPU intensive, but it looked validated to me. It was only upon trying out the “Write” button that I realized **Cell.toString()** was looping needlessly. I figured out that the loop didn’t matter and only the first character did. In fact, all it does is output either an “X” or “.” depending on the cell’s current state, so I just made it do that. Problem solved.

The first and last methods described here I used unit tests on. They were simple enough to do that for. However, the runContinuous() method was a void method, which was just one of several reasons why I found it difficult to test this way. It accomplishes several things all the while not returning a value. As a result, I decided to write manual tests for that method, as you can see on the following page.

One other point to note about the tests is that I made convertToInt() a public method so that it could be accessed by the JUnit tests.

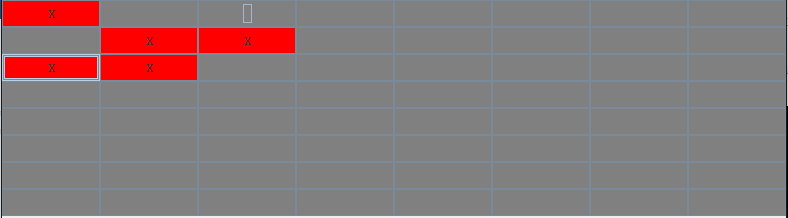
**Manual Tests** for MainPanel.runContinuous()

**Test 1**

TEST CASE: Make sure the “Run Continuous” button still ends up executing multiple steps ad infinitum

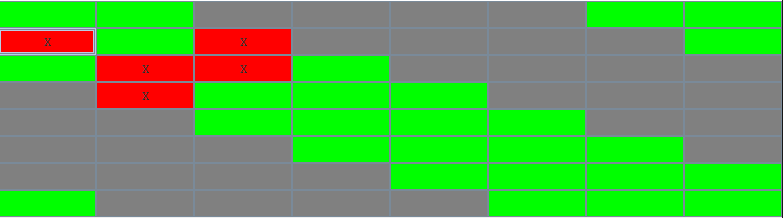
PRECONDITIONS: The board is loaded, clear, and stopped. It is of size 8x8.

EXECUTION STEPS: Create a “glider” as such:



Then press the “Run Continuous” button

POSTCONDITIONS: The “glider” should move diagonally down and to the right in accordance to the game’s rules, wrapping around when it gets to the edges. It continues to go over its same path without ever stopping on its own. That path is shown below in a screenshot at a randomly chosen point during execution.



**Test 2**

TEST CASE: Ensure that “Running…” continues to be outputted in the console

PRECONDITIONS: The board is loaded, clear, and stopped. It is of size 8x8.

EXECUTION STEPS: Press the “Run Continuous” button

POSTCONDITIONS: “Running…” should repeatedly appear in the console (more precisely, between every iteration calculation within the loop). This should not stop on its own.

**Test 3**

TEST CASE: Ensure that backup() is being called and executed repeatedly

PRECONDITIONS: The board is loaded, clear, and stopped. It is of size 8x8. The backup() method has been modified to call debugPrint() at the end.

EXECUTION STEPS: Create a glider. This can be done the same way shown in Test 1. Press the “Run Continuous” button.

POSTCONDITIONS: A display of the cells is repeatedly shown within the console. Each iteration within the console differs by one turn in accordance with the game rules. This should match what is shown by the actual application on any given iteration.

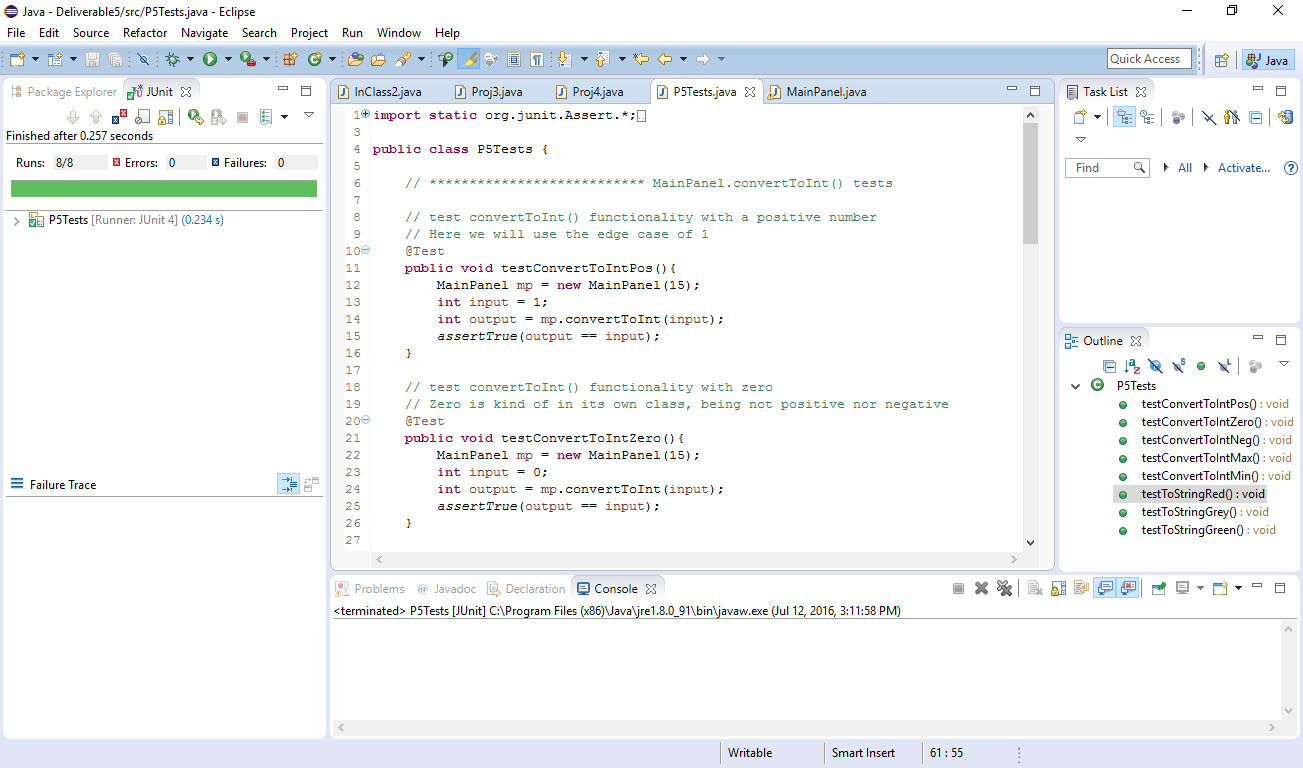
**Test 4**

TEST CASE: Ensure that calculateNextIteration() is being called and executed repeatedly

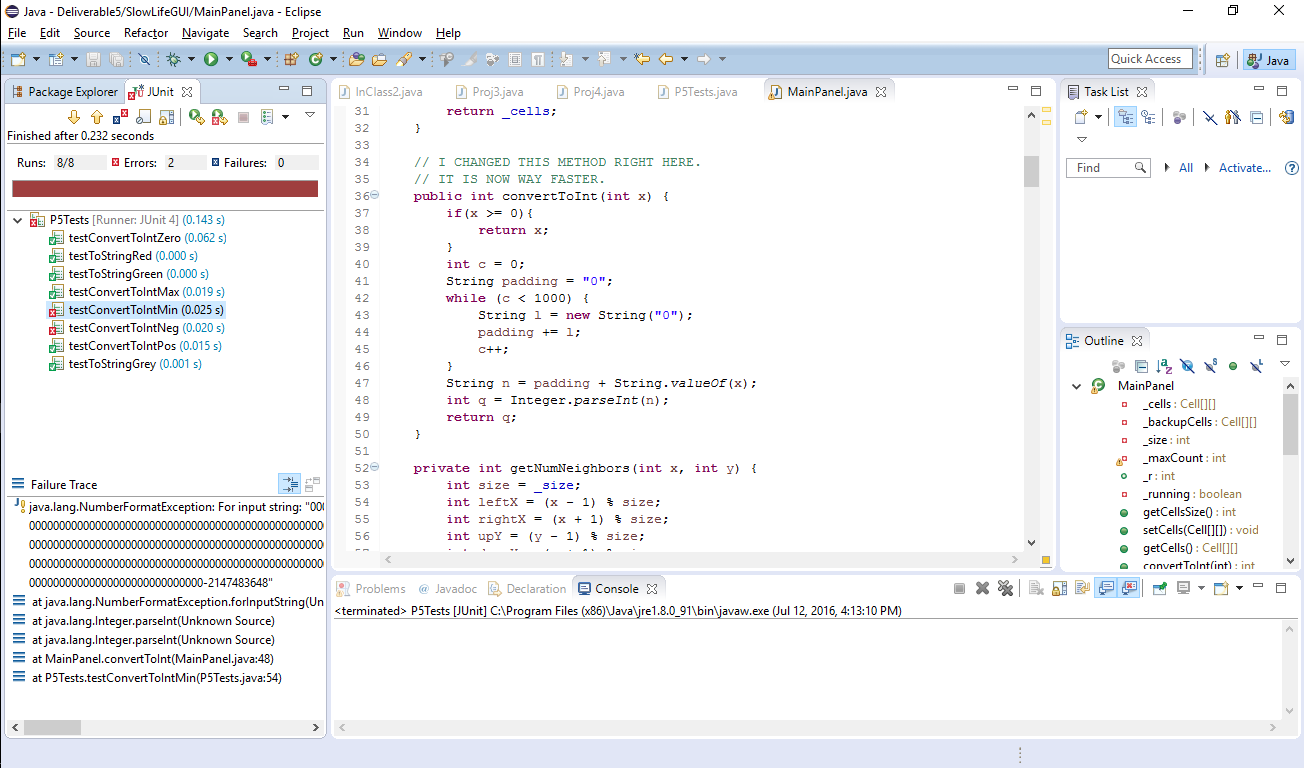
PRECONDITIONS: The board is loaded, clear, and stopped. It is of size 8x8.

EXECUTION STEPS: Create a glider. This can be done the same way shown in Test 1. Press the “Run Continuous” button.

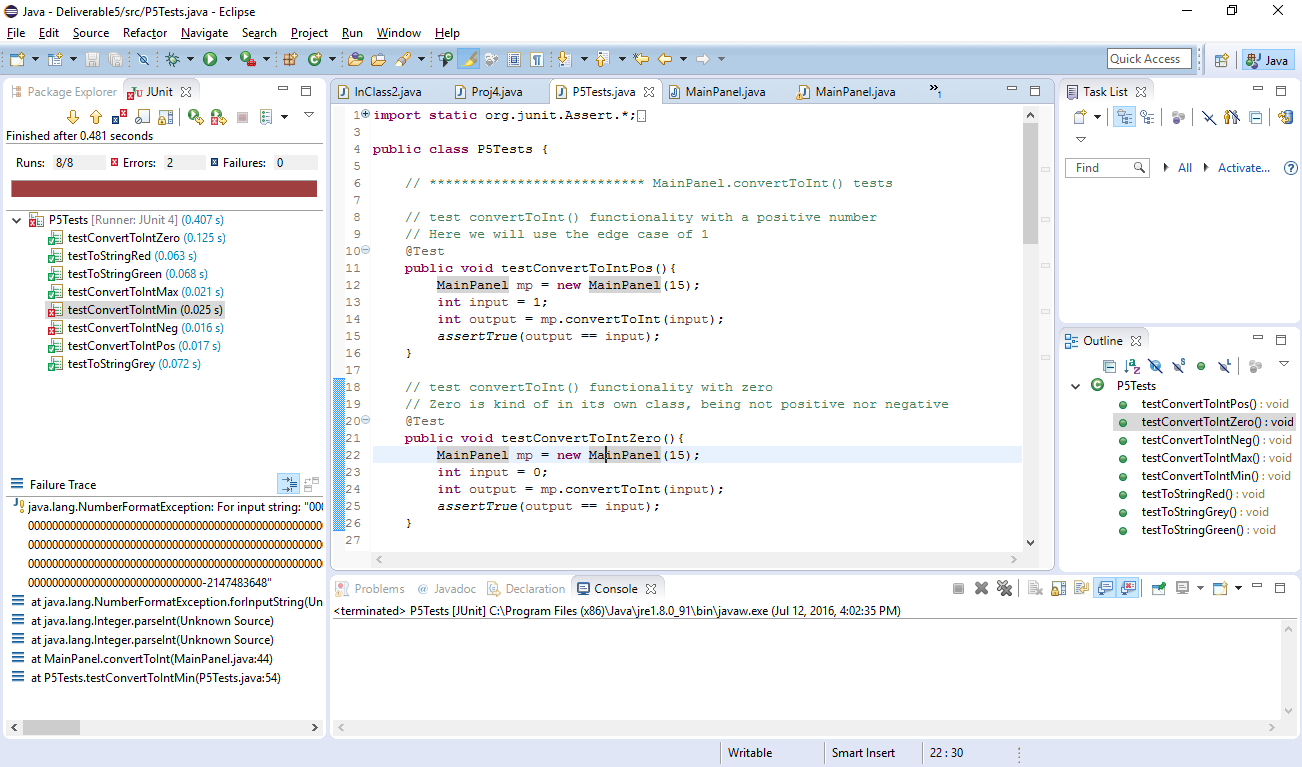
POSTCONDITIONS: In the console there is shown “Calculating..” and “Displaying…”, each on their own lines. This should be re-displayed on each loop iteration (if Test 2 has passed, you should see these outputs between “Running…” being continuously displayed as well).

My changed code (convertToInt() error fixed)

My changed code (convertToInt() error left to remain)



The old code



Note that in the last two screenshots, the same 2 tests have failed for the same reason, showing the lack of any change whatsoever (good or bad) in functionality.