CS 1632 - DELIVERABLE 4: Performance Testing Using VisualVM

Josh Demusz – joshdemusz

Github: https://github.com/joshdemusz/CS1632-D4

Summary

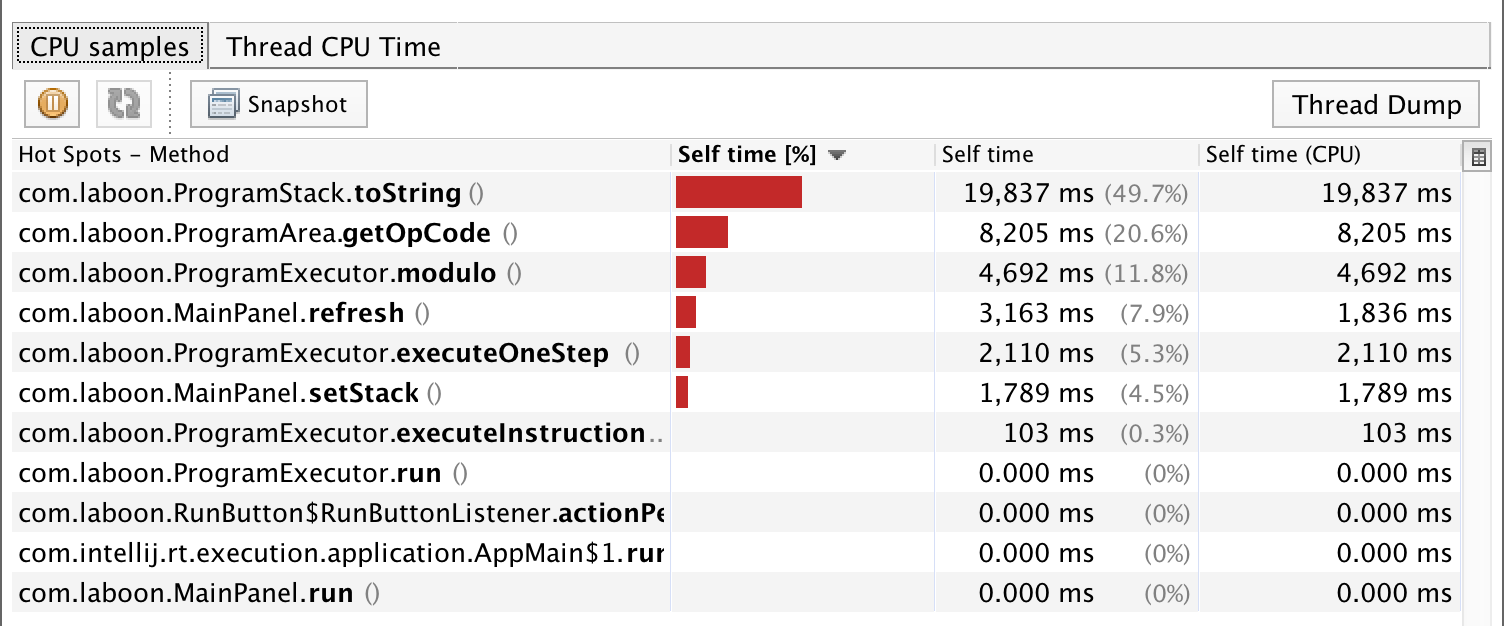
I used the VisualVM plugin for the Intelij IDE in order to do my profiling. In profiling the application, I ran the FizzBuzz program and checked to see the three methods that took up the most CPU time. I determined these methods to be toString(), getOpcode(), and modulo().

In order to refactor, I looked at the code in order to determine whether there was unnecessary code, or if the algorithms in the methods could be written more efficiently. Both cases were discovered in the method. In toString() and modulo() for example, both methods contained extra code that greatly increased the runtime of the method. This extra code could be deleted since it had no impact on the functionality of the program. In getOpcode(), a nested for-loop was used for a task that indexing into a 2D array could solve. Thus, using indexing instead of a nested for-loop greatly enhanced the efficiency of this method.

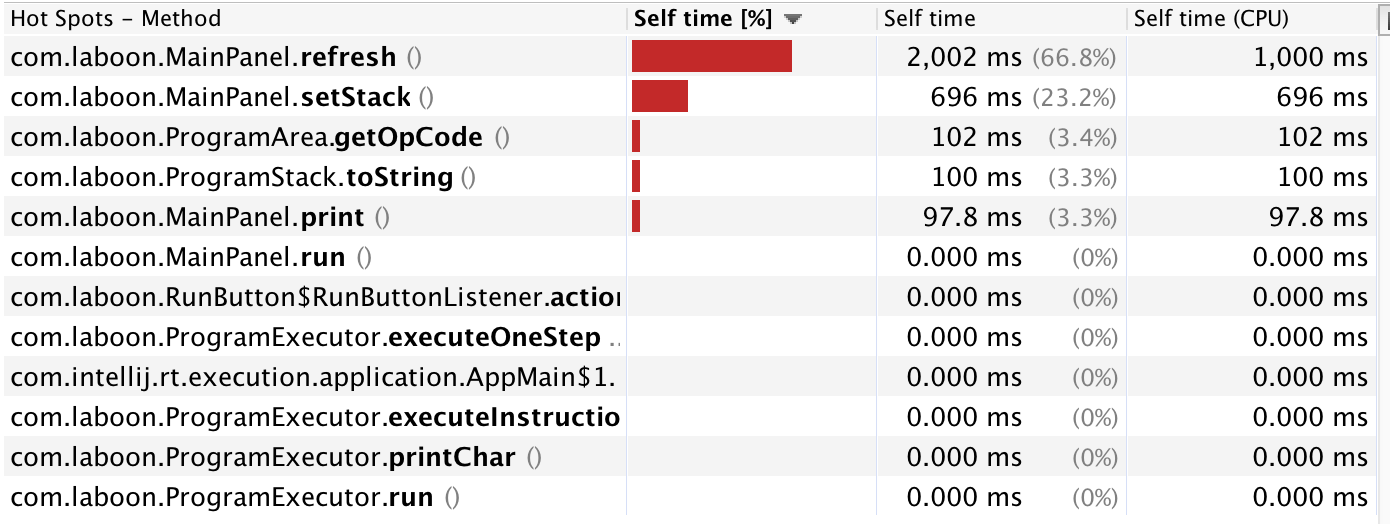
Assuming that the pinning tests for all methods passed after the refactor, I determined if the refactor was beneficial by comparing the initial run times with the current runtimes. I ran VisualVM again with the FizzBuzz program, and saw that the CPU times of the three methods I refactored greatly decreased. Thus, the refactors were beneficial to the performance of the application.

VisualVM Screenshots

Before refactor:



After refactor:



CPU Times

Before refactor:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trial 1 (ms) | Trial 2 (ms) | Trial 3 (ms) | Average (ms) |
| toString() | 19,873 | 20,089 | 19,067 | 19676.333 |
| getOpcode() | 8,205 | 8,794 | 6,896 | 7965 |
| Modulo() | 4,692 | 4,474 | 5,158 | 4774.667 |

After refactor:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Trial 1 (ms) | Trial 2 (ms) | Trial 3 (ms) | Average (ms) |
| toString() | 100 | 98.8 | ~0 | 66.267 |
| getOpcode() | 102 | ~0 | ~0 | 34 |
| Modulo() | ~0 | ~0 | ~0 | ~0 |