2015

CS 1632 – DELIVERABLE 5

PROFILING/REFACTORING

LOGAN KAUSCH

To profile the application, I used VisualVM and ran the code three separate times, varying a few arguments between runs. The first run was of size 15 with 20% of the cells initially being alive. The second run was of size 15, as well, but with 40% of the cells initially being alive. Lastly, I ran the code with a size of 20 and 30% of the cells were initially alive. When I profiled these with VisualVM, I noted that each run gave a similar indictment of the application. The World.java code seemed to be taking up roughly 60% of the CPU allotted for the program (specifically the Iterate method), as seen in Figures 1-3. Therefore, this was the code that was to be refactored.

When I looked at the Iterate method within World.java, I realized that it was not likely the problematic method, as its only purpose was to instantiate new Cell.java instances, calling GetNumNeighbors in the process. When I looked at GetNumNeighbors, it was obvious that it was the problematic method, as it contained a suspicious for-loop that ran a selection of if-statements 10000 times. Upon further inspection, I realized it was merely checking to see if a neighbor was out-of-bounds, replacing any -1s with size-1. However, knowing that moving the right or down would never result in -1, I removed that code from the loop. The use of a mod (%) in the initialization of the right and down x and y values, respectively, removed the need for those checks. Other than removing the mods from the initialization of the left and up x and y values, respectively, I removed the loop entirely from the method.

Running the same three instances of the code again and profiling them with VisualVM proved that this was the issue. The World class was no longer taking up an inordinate amount of CPU usage and all of the usage was going towards running the main method in JavaLife.java, as seen in Figures 4-6.

GitHub Link: https://github.com/lmk65/JavaLife

Figures

Figure 1. Before Refactoring, Size 15, 20% Initially Alive

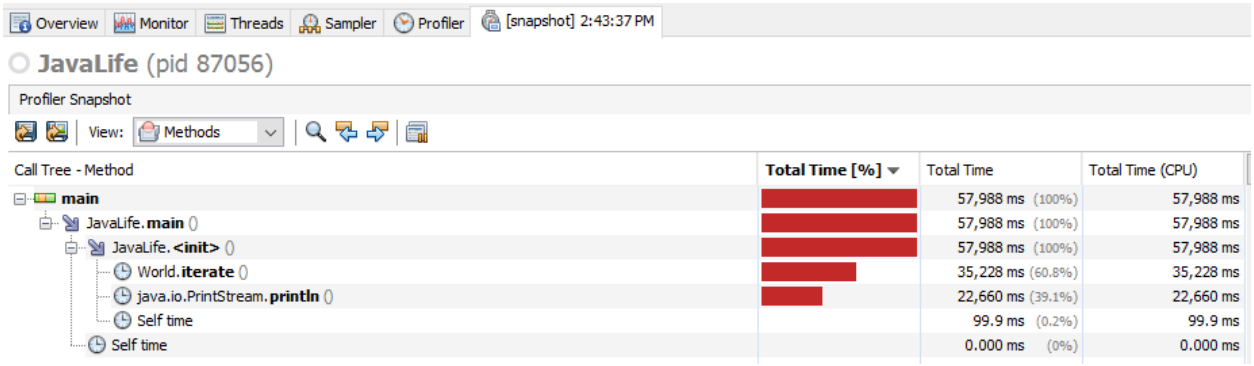

Figure 2. Before Refactoring, Size 15, 40% Initially Alive

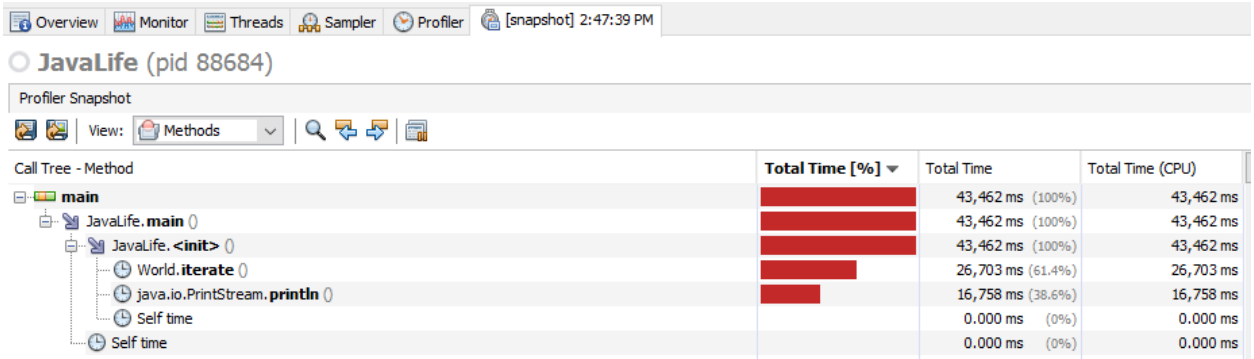

Figure 3. Before Refactoring, Size 20, 30% Initially Alive

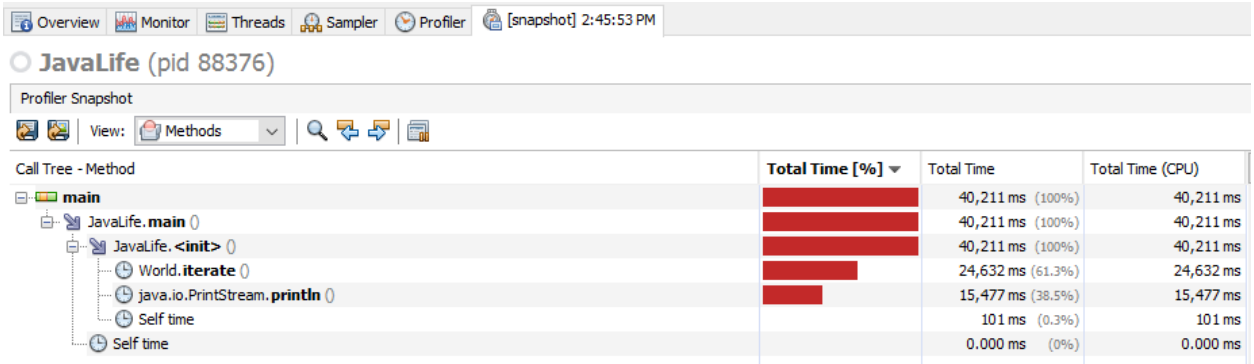

Figure 4. After Refactoring, Size 15, 20% Initially Alive

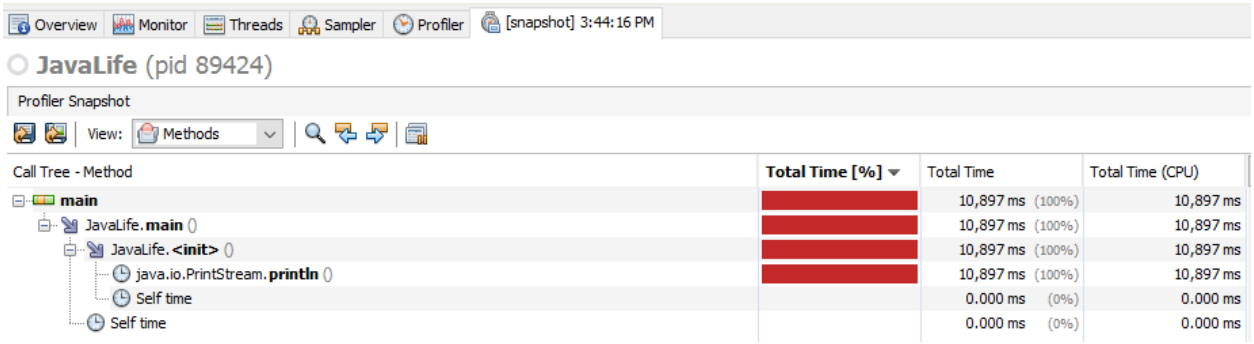

Figure 5. After Refactoring, Size 15, 40% Initially Alive

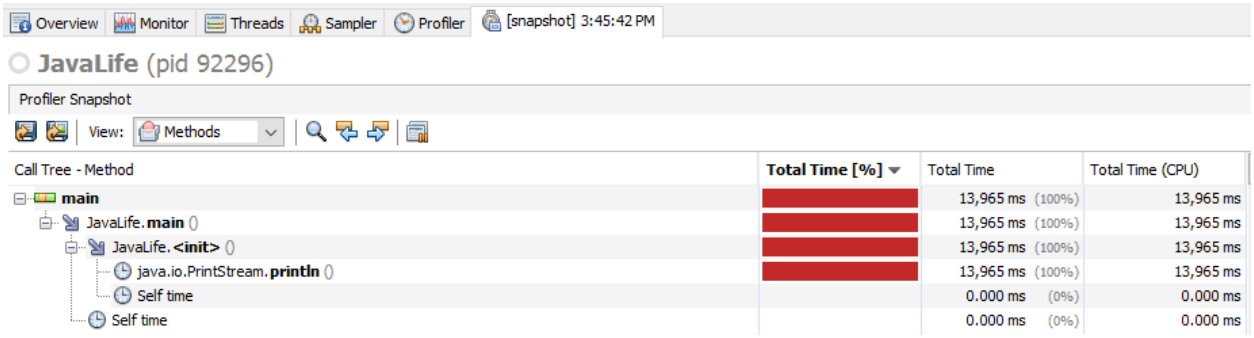

Figure 6. After Refactoring, Size 20, 30% Initially Alive