**James Hantz (jeh170), Karnbir Singh (KarnbirSingh)**

**https://github.com/jeh170/CS1632\_deliverable4**

**CS 1632 – DELIVERABLE 4: Performance Testing Using VisualVM**

**KPI’s:**

1. When running *fizzbuzz* on JBefunge using the “run” button, we do not want to exceed a runtime of 1 second.
2. When running *fizzbuzz* on JBefunge using “run”, we do not want to exceed a runtime of .5 seconds for the program area to display the results.
3. When running *fizzbuzz* on JBefunge, we do not want to exceed a runtime of .5 seconds for the cursor highlighter.

**Summary:**

In order to determine the first KPI, *“When running fizzbuzz on JBefunge, we do not want to exceed a runtime of 1 second,”* we sought to ensure that the *run* operation would work with a high level of efficiency, since the user does not require any real-time feedback from the system during execution. Unlike other modes of functionality, such as *walk* or *mosey*, *run*’s task is to complete the program, and display the results as quickly as possible.

Testing the first KPI required us to obtain a VisualVM profile of JBefunge executing *fizzbuzz* on *run* mode. The results showed that it took longer than 1 second for the entire runtime. As a result, we examined the *run()* method in the *ProgramExecutor.java* file. Here, we learned that when *run()* is invoked by user pressing *run* on to execute *fizzbuzz* code, *highlightChar(), setStack(),* and *refresh()* methods are invoked for each opcode execution. If we can avoid invoking these methods for each individual opcode instruction (until stopped), then we can increase the runtime.

The *run()* method was refactored by, ensuring that *setStack()* and *refresh()* only need to be invoked once when the files reaches the stop opcode. We do this by moving these methods outside of the while-loop that runs the program until stop-opcode is found. As a result, we found a dramatic improvement in performance run-time. Additionally, we removed the *highlightChar()* invocation, as it is not necessary for the user to observer opcode characters being highlighted while *run* is executed.

Another KPI, we sought to achieve was making sure the program was only executing such that it would not continually update the program area, hence taking up runtime. It was visually clear that the program was taking up additional resources/time just to print everything as it was running. As a result, we explored the *refresh()* method, which was also being called each time an opcode instruction was executed. As a result, we moved the *refresh()* method so that it would only be invoked just once, to display the final output.

The third KPI is to improve performance by making preventing the JBefunge from unnecessarily highlighting opcodes when running. This is because the user has no concern of the program execution when in run mode. The main target is to obtain the final output. As a result, we changed the *run()* method to ensure that it does not highlight any characters if it is in run mode. We also changed the *highlightchar()* method in the MainPanel class, as it was instantiating a new *DefaultHighlighterPainter* object every time it was called. This method was changed so that a new instance of a *DefaultHighlighterPainter* would only be created when the color of the cursor was changed. This was done in an attempt to improve speed, but may have improved memory usage much more than speed.

**Before and After VisualVM Screenshots:**

**Initial and Final Measurements:**