**Project Under Test: MonkeySim**

Ritika Maknoor

David Bickford

10 November 2016

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

**SUMMARY**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

When we began this deliverable, we knew that our end goal was to ultimately improve the performance of our project under test: MonkeySim. In order to accomplish this, we had to first determine which aspects of the program were responsible for the hindrance in performance. Measuring performance, in this case, was part of efficiency-oriented resource utilization. By measuring the utilization (what percent of the computing resources were being used to perform the tasks/run the methods in the program), we would be provided with a detailed logging of the specific methods’ CPU usage. This was done through a program-specific profiler tool: VisualVM.

Through VisualVM, we were able to run the application (with or without an argument also being passed in), and get a visual recording of the CPU usage of the different methods in the program- in the “Sampler/CPU” tab. We noticed that 3 methods in particular seemed to be taking the longest amount of CPU time to run (in this order of most to least):

1. MonkeySim.stringifyResults()
2. Monkey.generateId()
3. MonkeySim.getFirstMonkey()

MonkeySim.stringifyResults()

We decided to try and refactor stringifyResults() first because it was using the most time/affecting performance the most. We examined this code and saw that logically, some of the code/work was not needed to be done in order to complete the task it was meant to do, which was to return a string version of the correctly formatted output of a specific round. We noticed that in the try block, before it actually constructed the string to later be returned, it concatenated 50,000 ‘@’ symbols to the string toReturn. Also, after the try/catch block, when it was time to return the resulting string, it returned a subset of the resulting toReturn string (50,000 ‘@’ symbols and the correctly formatted output at the end of it). By calling .substring(HEADER) on the toReturn string before returning it, it removed all of the 50,000 ‘@’ symbols at the beginning of the string, and ultimately just returned the correctly formatted subset. We saw all of this work being done in this method to be extremely unnecessary. We decided to refactor this method by both getting rid of the initial for loop that concatenated the ‘@’ symbols, and getting rid of the .substring call before returning. We then were left with a toReturn string that was ever only what we wanted to return from this method.

Monkey.generateId()

We then decided to try and refactor generateId() next. We knew that the purpose of this method was simply to take in an int as monkey number, and return that Monkey’s ID int value which always stays constant. We noticed that the algorithm for how an ID was assigned from a monkey number was by simply adding 223492 to it. Therefore, we refactored this method to merely add 223492 to the int n it took in as a parameter, and return it.

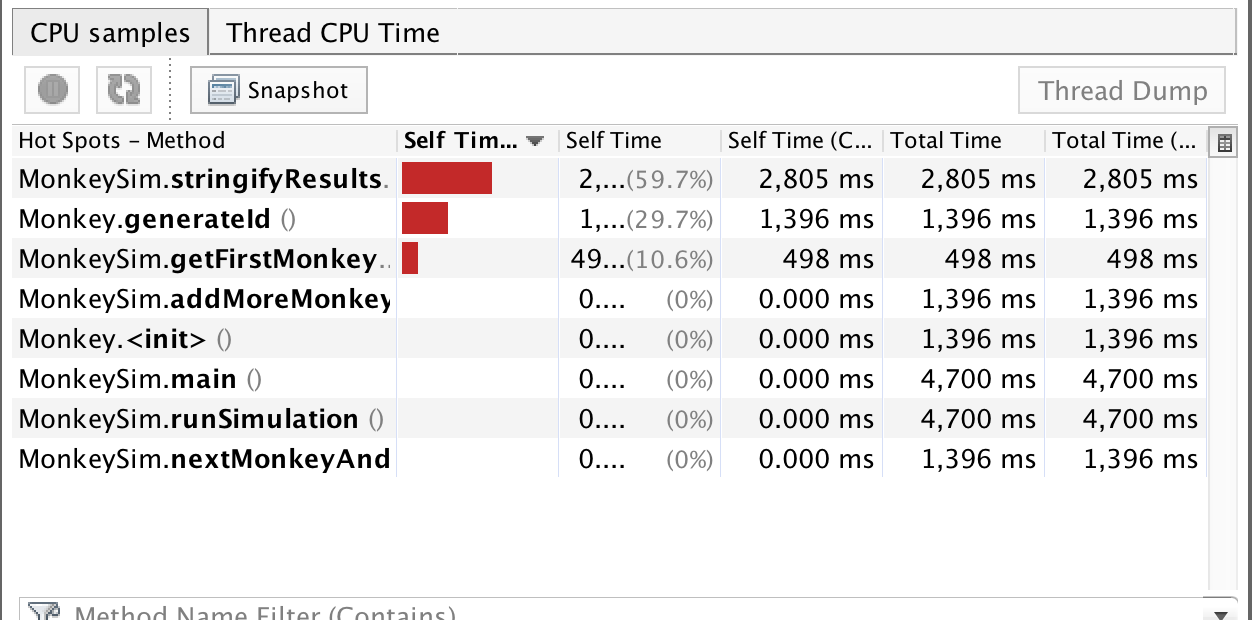
MonkeySim.getFirstMonkey()

We then decided to try and refactor getFirstMonkey() next. We knew that the purpose of this method was to get a reference to the first monkey in the list. Through examination of the code, we noticed that by using a LinkedList data structure, it was starting from the end of the list and traversing to the beginning each time it was called so that it could return a ref to the first monkey. We refactored this by just starting at the beginning of the list each time instead.

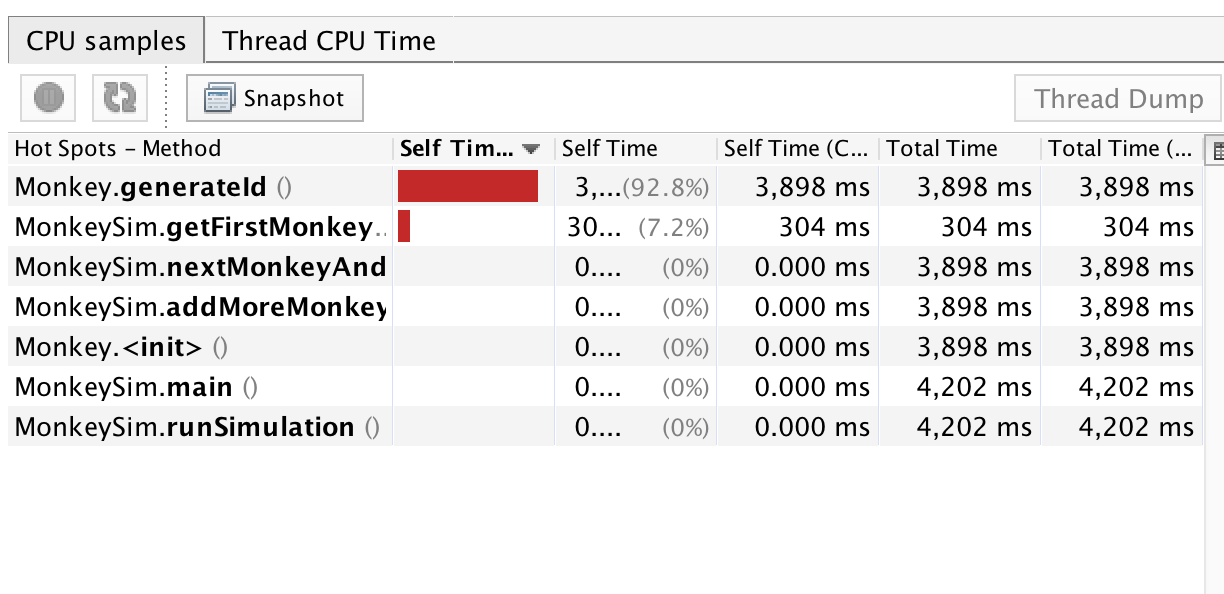
**SCREENSHOTS OF VISUALVM**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

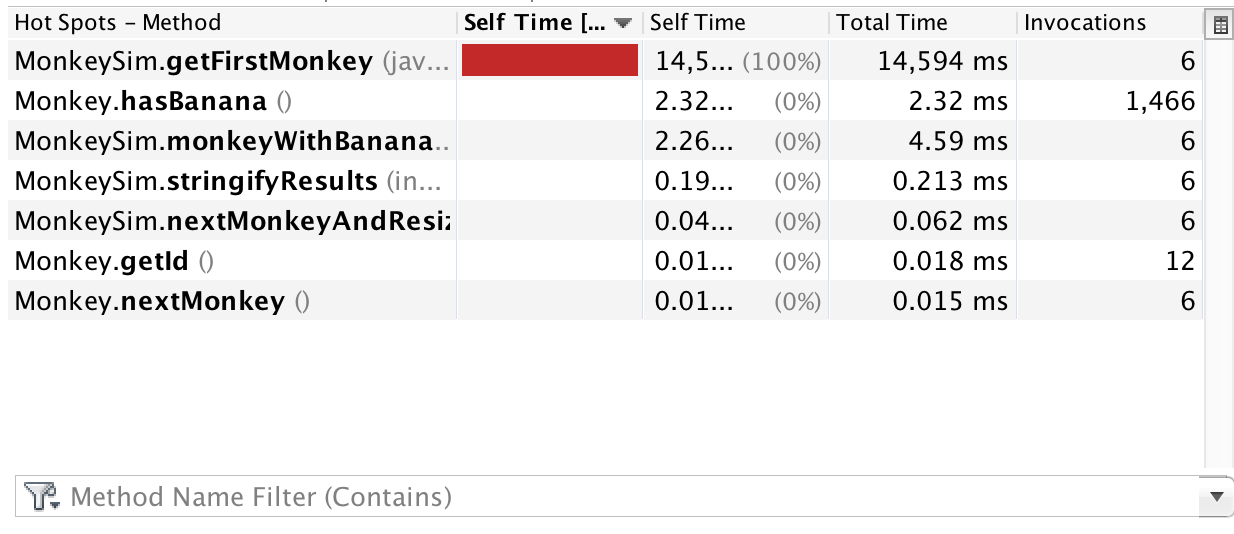
**BEFORE any refactoring:**

****

**AFTER refactoring stringifyResults()**



**AFTER refactoring generateId()**



**AFTER refactoring getFirstMonkey()**

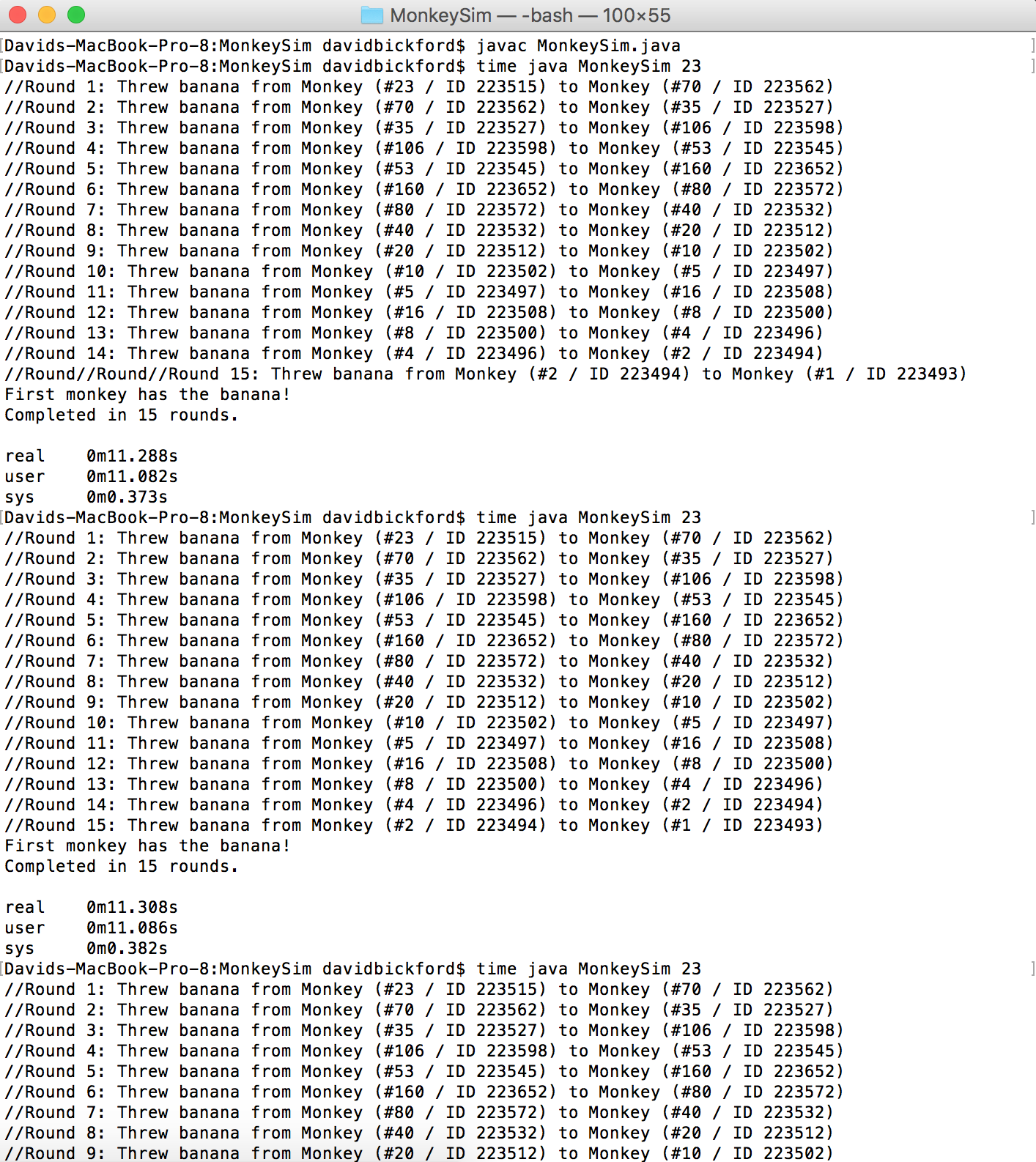
Was too fast to be able to run program and open profiler in time for a screenshot.

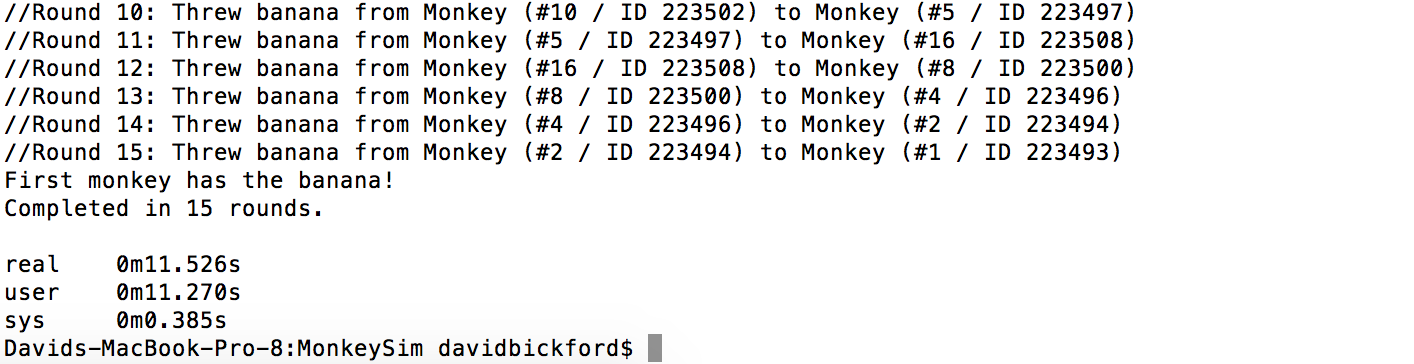
**INITIAL & FINAL TIMES**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**BEFORE any refactoring**

**Initial monkey: 23**



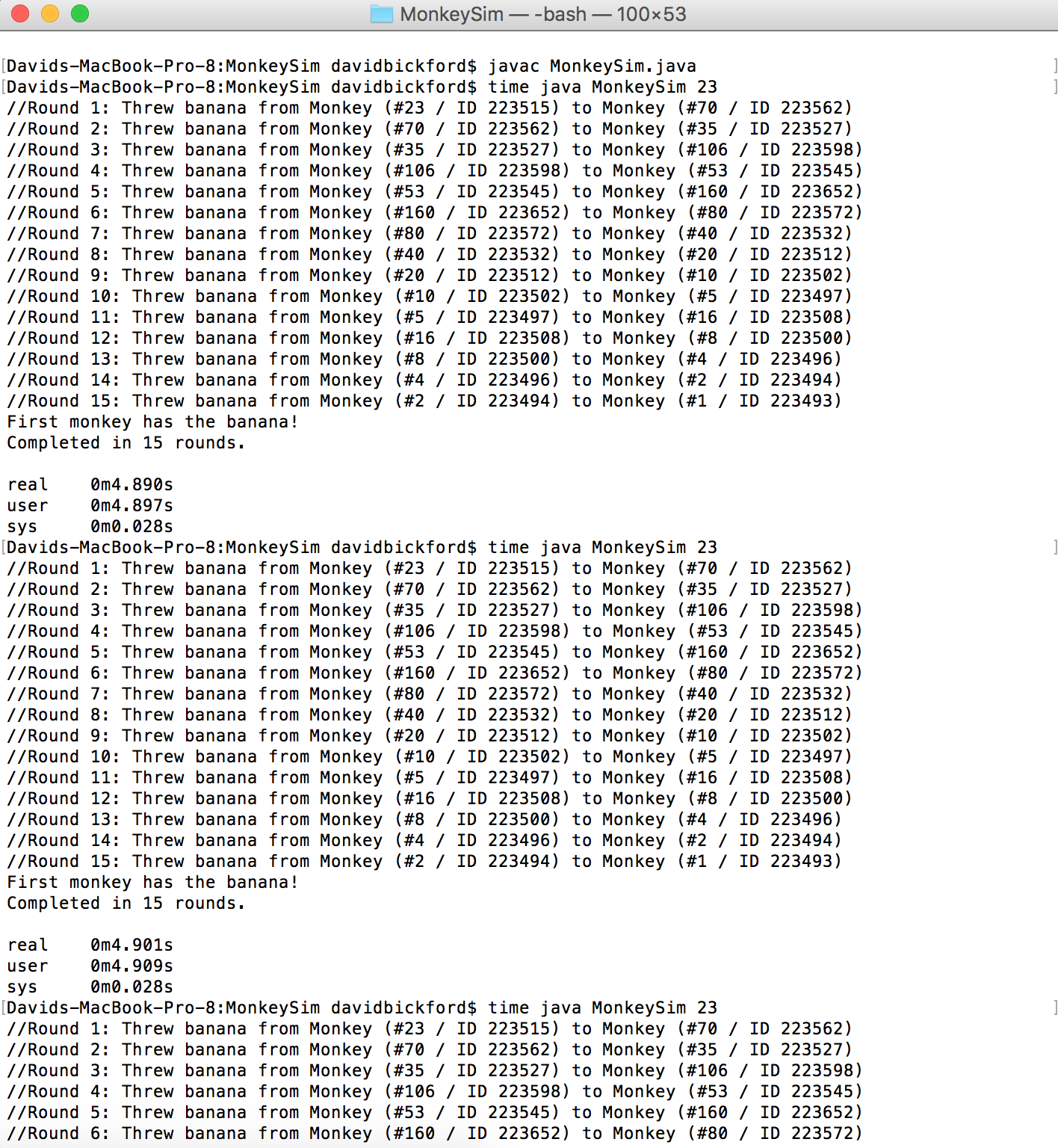


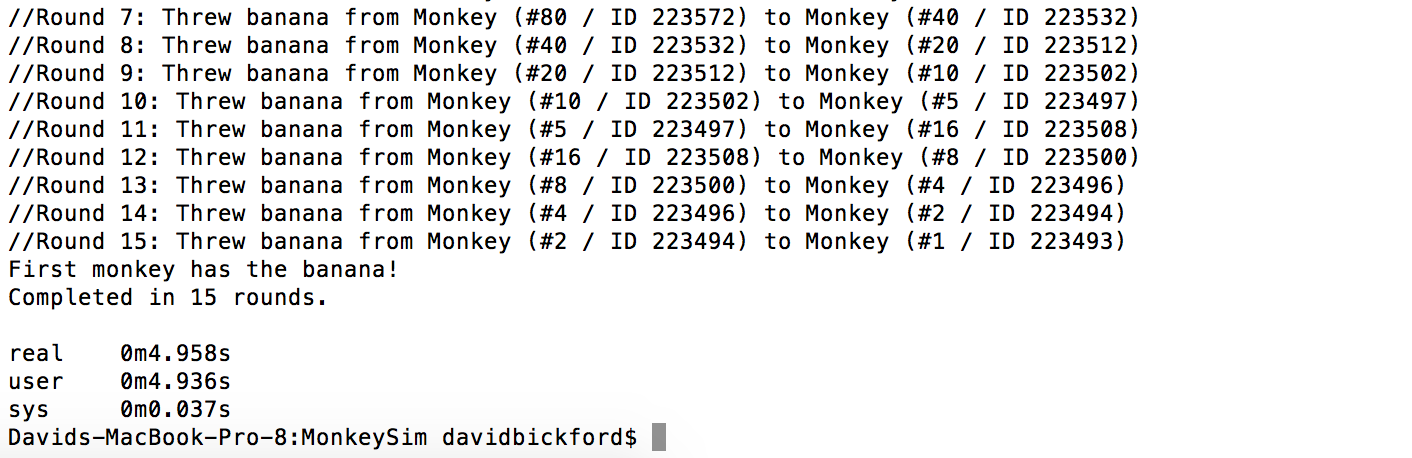
* Max amount of real time: 0m11.526s
* Mean time of real time: 0m11.374s

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**AFTER refactoring stringifyResults()**

**Initial monkey: 23**



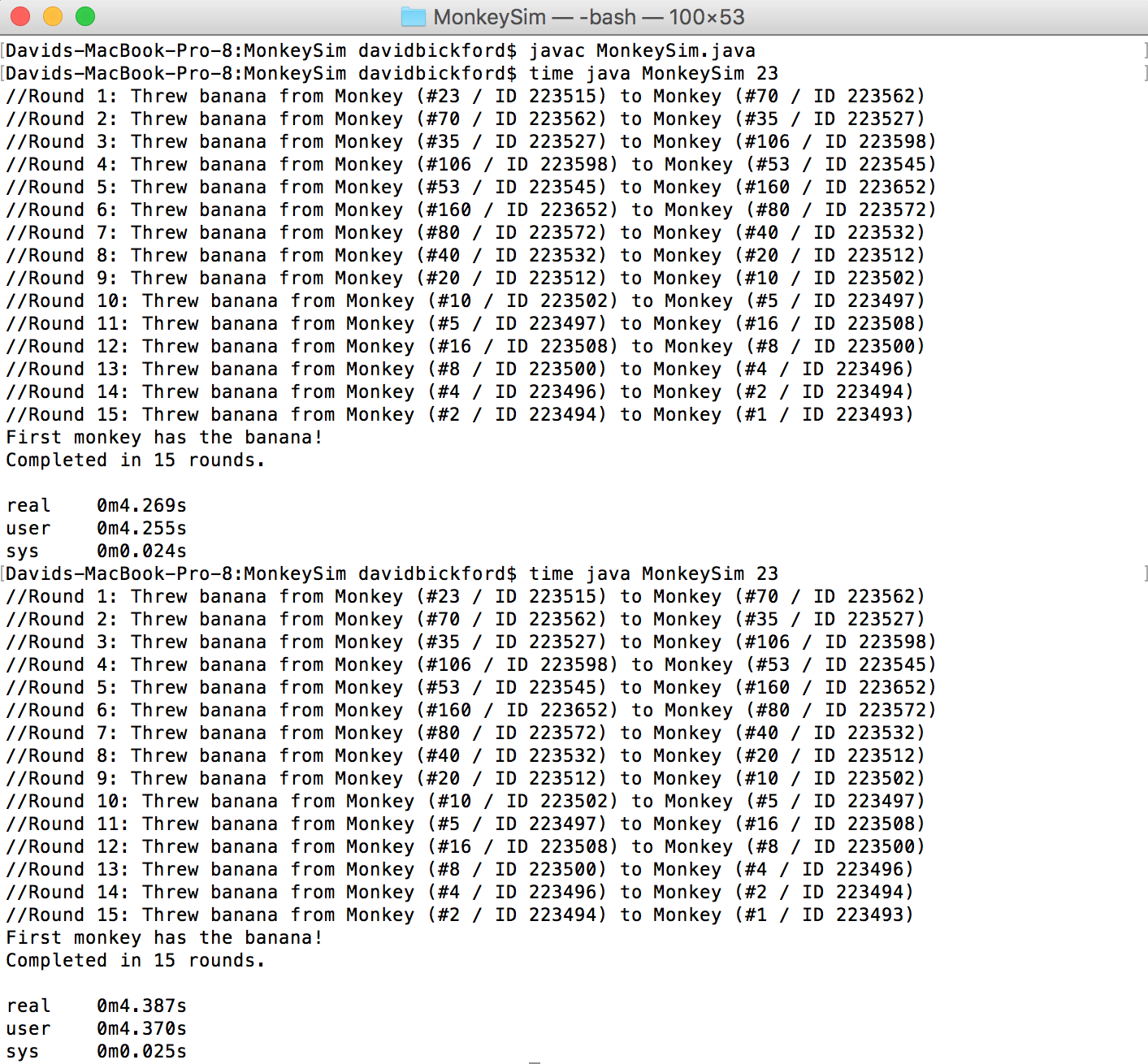


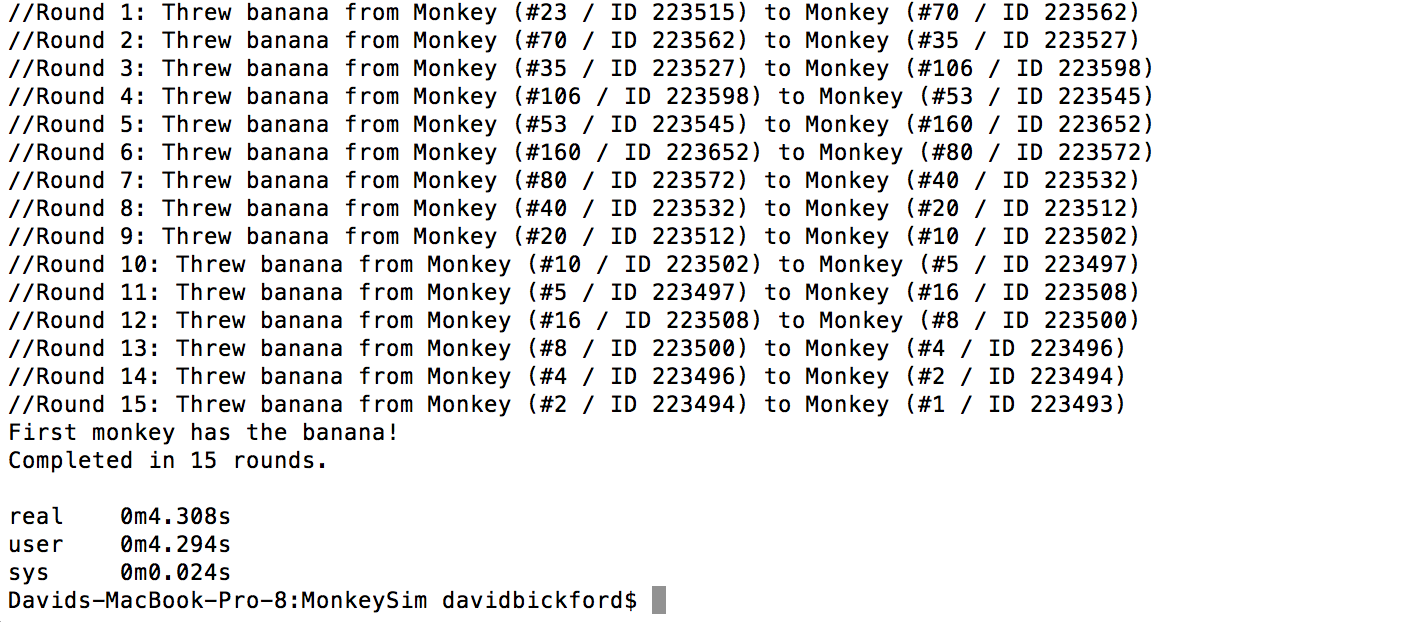
* Max amount of real time: 0m4.958s
* Mean time of real time: 0m4.916s

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**AFTER refactoring getFirstMonkey()**

**Initial monkey: 4**



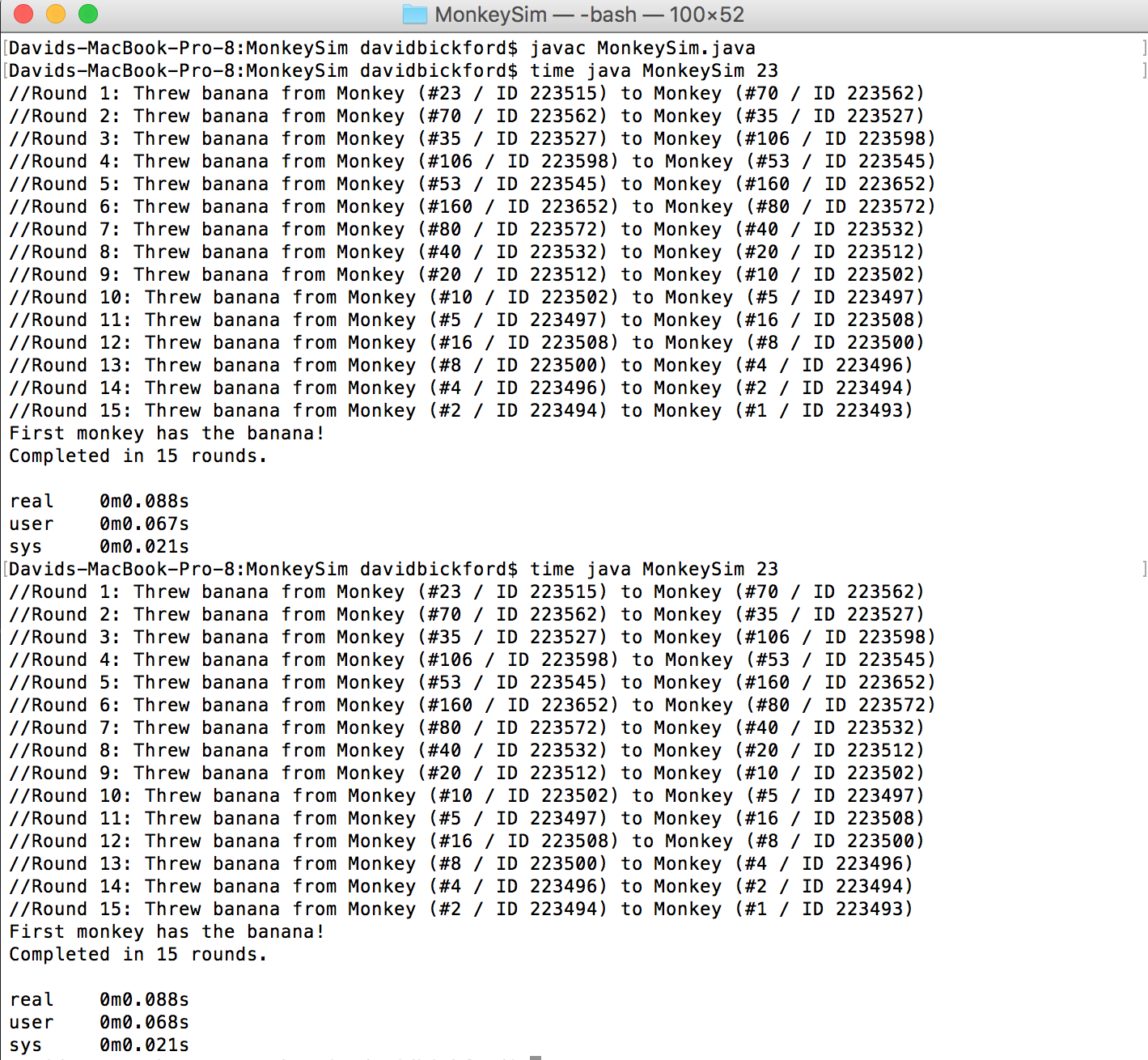


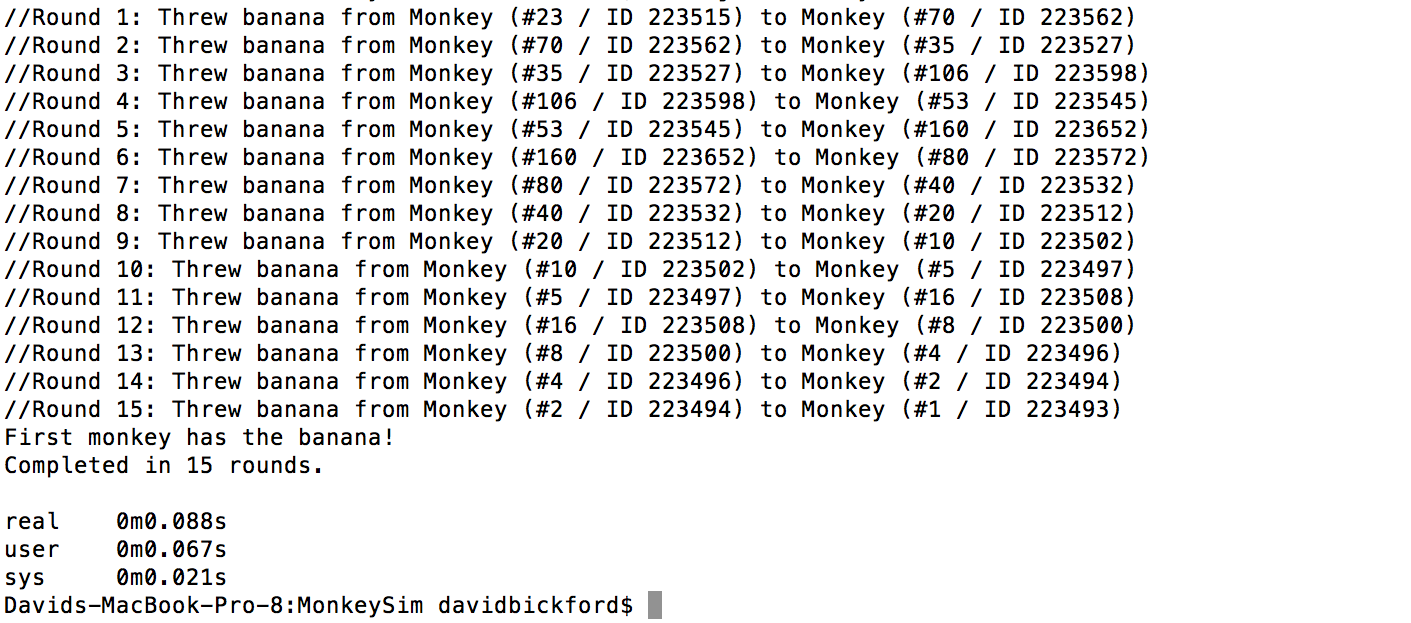
* Max amount of real time: 0m4.387s
* Mean time of real time: 0m4.321s

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**AFTER refactoring generateId()**

**Initial monkey: 4**





* Max amount of real time: 0m0.088s
* Mean time of real time: 0m0.088s

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_