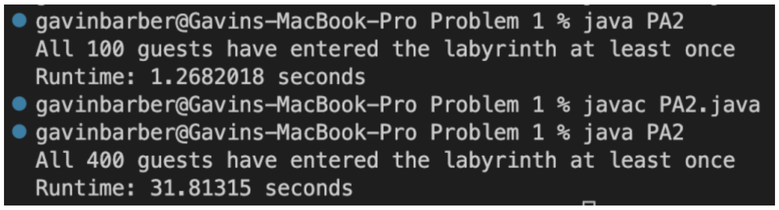
Assignment 2 Experimental Evaluation

Problem 1: Minotaur's Birthday Party

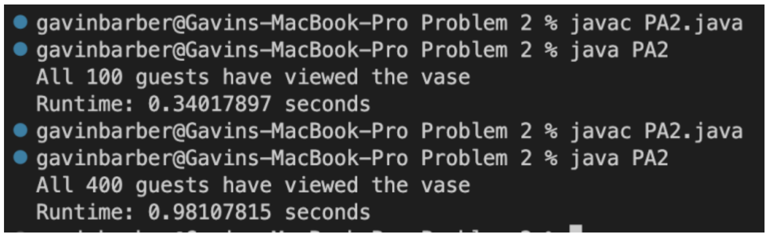
I revised my approach a couple times before settling on what works best. The first thread created will always be the "counter" and the rest will follow the same set of rules. The counter's job is to check if a cupcake is missing, if so, request a new cupcake, and increment the count. The rest of the threads, including the counter, are to check if they have eaten a cupcake and if a cupcake is present. If conditions are correct, then they may eat the cupcake. Once the counter's count variable is equal to the number of threads/guests then we know that all guests have entered the labyrinth at least once.

I ran into many synchronization issues while working on my program. My biggest issue is that my program will randomly get stuck during execution and other times finish execution successfully. Luckily, I got it to more likely finish execution correctly than not. I believe the issue has to do with some deadlock or race condition problem. I did hours of debugging and research but could not get it fully correct. When it does run to completion the runtime is not the best and it grows fast. I noticed a huge jump in runtime when I tested 100 guests versus 400 guests.


```
● gavinbarber@Gavins-MacBook-Pro Problem 1 % java PA2
  All 100 guests have entered the labyrinth at least once
  Runtime: 1.2682018 seconds
● gavinbarber@Gavins-MacBook-Pro Problem 1 % javac PA2.java
● gavinbarber@Gavins-MacBook-Pro Problem 1 % java PA2
  All 400 guests have entered the labyrinth at least once
  Runtime: 31.81315 seconds
```

Problem 2: Minotaur's Crystal Vase

My approach to this problem was very straight forward and much smoother after struggling with problem one. I chose to go with the second option of having a "BUSY" or "AVAILABLE" sign. Each thread will check the status of the sign until they are able to enter. I added a very short sleep function once a thread enters the labyrinth to simulate viewing the vase. I ran this test with 100 to 400 threads and both times ran under a second. This program is much faster than problem 1 because it does not rely on randomness.


```
● gavinbarber@Gavins-MacBook-Pro Problem 2 % javac PA2.java
● gavinbarber@Gavins-MacBook-Pro Problem 2 % java PA2
  All 100 guests have viewed the vase
  Runtime: 0.34017897 seconds
● gavinbarber@Gavins-MacBook-Pro Problem 2 % javac PA2.java
● gavinbarber@Gavins-MacBook-Pro Problem 2 % java PA2
  All 400 guests have viewed the vase
  Runtime: 0.98107815 seconds
```