Ellis Fitzgerald
COS 420: Object Oriented Design
March 31, 2025
AI Lab Book #5

LLM: DeepSeek DeepThink R1
Time Spent Overall: 1 hour & 30 minutes
Time Spent Prompting: 20 minutes

Build a First Singleton

For the start of this assignment, the organization of my class and objects lacked encapsulation. Other than the players and the dice, my whole program was in the "main" file. This required some refactoring of sorts to make the referee possible. Despite us creating a Model to abstract our ArrayList of Players and their scoring in the previous assignment, there are some conflicting instructions regarding having the referee take an "ArrayList" of Players as a method parameter. To keep things consistent, I decided on keeping the Model abstraction for the referee class.

The way DeepSeek implemented the Singleton was close to how we learned in lecture. A static variable in the class definition associated with a getInstance() method which checks its nullity before maybe lazily initializing and always returning. However, there was one thing that was rather inconsistent or perhaps more confusing to deal with: dependencies. The referee can not necessarily act on its own. In order for things to show up in the view (our GUI) it needs to signal events of some sort. This resulted in a situation where getInstance() (as expected) takes no parameters. However, this meant it was the programmers job to make sure they *eventually* called other functions that initialized the GameEventListener (an interface DeepSeek created earlier on in the semester to signal certain events for the GUI) and the Model. I personally did not like this approach as it leads to what could be an easy programmer error. There is no syntax requiring the initialization of these two objects, forcing the hand of an awkward solution which DeepSeek opted for by writing a method "validateState()" which checked the nullity of the objects and threw an exception upon success. I simply changed this so methods like startGame() require parameters "Model" and "Listener." This technically is in line with the instructions requiring us to pass an ArrayList of players. My solution is not the most elegant considering these references needed to be chained and passed to other functions within the referee class. Admittedly, this would be easier to implement given DeepSeek had not opted for the GameEventListener interface in the first part, or if the startGame and startNextTurn methods were combined into an iterative approach instead. It brings up the question: if most (or all) of the references and data accessed by an object/singleton via method parameters, is there really a point to it even being an object? Technically, it could be just static functions. I wanted to revisit and revise this part a few times as it felt a little unfaithful to what was asked, but I could see arguments for both sides.

The other thing that differed from our lecture implementation of Singleton was the use of the “synchronized” reserved word. Doing some light research explains how this keyword blocks other threads from accessing the same function at the same time. As far as I can understand, this is because our Java Swing program is not on the same thread(?) It should be known that it is not a mutex, it is actually more accurately described as a monitor which I found interesting.

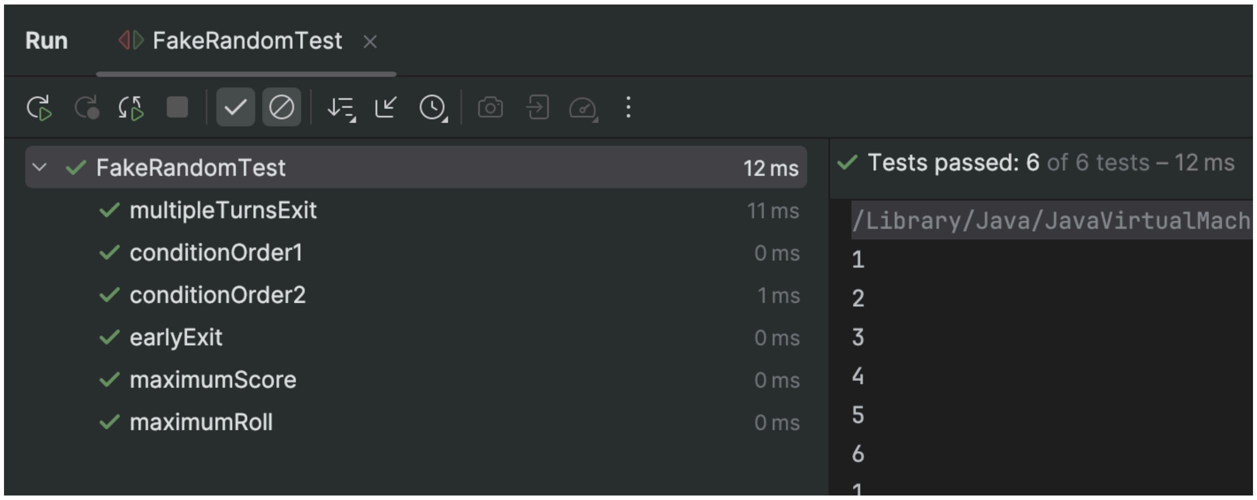
Unfortunately, implementing the new referee class into the main program was not easy. I initially wanted to see if DeepSeek could do it with some guidance with my new prompting strategies. However, after a few prompts where I gave the entire main file to DeepSeek it seemed to forget a lot of functionalities. One example is the dialog buttons used by HumanPlayers no longer had listening functions, DeepSeek seemed to not care about having a “WINNING_SCORE” constant, and turns would not properly cycle. On another note, although it is not a functional issue, DeepSeek is not great about commenting despite multiple reminders about using JavaDoc. Admittedly, the instructions somewhat hinted at myself manually moving over a few functions over to the Referee class. Perhaps if I started there the replacement would have been much easier. Nonetheless, I chose for the LLM to do one part, and I do the other which seems at least partially in line with the shared responsibilities.

Due to the issues of hallucinations and DeepSeek forgetting so much, I decided to simply comment out the “startNextTurn” functions, convert any GUI modifications into listening functions, and replace any gameflow function calls to the Referee singleton. This way I could keep my same Main program with the same JavaDoc and it could actually work.

How the LLM Performed

Summarized plainly, it seems it's much harder to trust an LLM with doing a “moving job.” As soon as you have it start fresh is where its application seems to get increasingly useful. Though, I would argue that this assignment was more moving than it was a generation. Especially for something like a Singleton where a lot of its implementation is really the getInstance function. I have noticed that if you give it too much of one thing (in my example of a project, it's the listener interface) and it often likes to keep spitting more of it out. Starting fresh and small scoped seems much more beneficial. In the future, I think I would opt for *not* pasting my entire file to try and refresh its memory. It seems that only adds to whatever dumpster fire I have.

Test Cases


Run FakeRandomTest

```
✓ FakeRandomTest            12 ms
    ✓ multipleTurnsExit     11 ms
    ✓ conditionOrder1        0 ms
    ✓ conditionOrder2        1 ms
    ✓ earlyExit              0 ms
    ✓ maximumScore           0 ms
    ✓ maximumRoll            0 ms
```

```
✓ Tests passed: 6 of 6 tests – 12 ms

/Library/Java/JavaVirtualMach
1
2
3
4
5
6
1
```

Figure 1: Success for all test cases.