Bulldog: Singletons and JUnit

Author:
Michael Yattaw

March 26, 2025

For this part of the program, I began by looking over my code and viewing which classes may need to utilize a referee. After about 10 minutes of reviewing my code, I had an idea of what to prompt the DeepSeek AI model with. The prompt I gave looked like this:

> I have a simple game written in Java; however, I want to create a new Referee class to handle the game logic instead of having a JPanel contain the game's logic. Can you create a Referee class referencing my JPanel class below?

This was then followed by the GamePlayPanel class. I then had to specify to the model that I did not want to deal with any JComponents at all in this Referee class, which then outputted a new Referee class that contains a GameEventListener interface. This prompting process took a total of 30 minutes to get correct overall.

```java
/**
 * Interface for game event callbacks
 */
public interface GameEventListener {
    void onGameStarted();
    void onTurnStarted(Player player);
    void onDiceRolled(Player player, int value);
    void onTurnEnded(Player player, boolean byChoice);
    void onAIPlayed(Player player, int score);
    void onScoresUpdated();
    void onGameEnded(Player winner);
}
```

I then had to implement this new interface inside the GamePlayPanel class, which took the DeepSeek model about 10 minutes to properly implement without errors, and I spent another 5 minutes testing to ensure the code worked properly.

Now the next step was for me to convert this new Referee class into a Singleton. So I prompted the AI with:

> Convert this Referee class into a singleton, and reference and update the code in both the GamePlayerPanel class and the BulldogReferee class pasted below.

This was followed by my two classes. The prompt took about 5 minutes to process by the DeepSeek model; however, the code executed perfectly fine, requiring minimal time for testing. Now, to compare the singleton with the lecture notes, I noticed that instead of a PlayerModel variable with getters and setters, the model decided to use a resetInstance method, which takes a player model parameter and would create a new instance of the singleton and update the player model variable. The DeepSeek model also ensured thread safety by using synchronization for creating new instances to prevent race conditions. However, resetting instances could be confusing, and I wanted a more pure singleton class, so I prompted the DeepSeek model with:

> Can you make the Referee class a pure singleton and update the player model variables?

Now the class looked a lot more similar to the lecture example, which converted the reset instance into a reset method and added a new method to update player models, which was now called inside the start game method, similar to how resetInstance functioned before.

The next part of the program was to create a RandomDice superclass and a FakeRandom dice class that takes in fixed dice values. The FakeRandom class was created with this prompt:

> Construct a FakeRandom subclass of RandomDice that uses a stream of data. FakeRandom should implement the roll method, which returns the next item.

The DeepSeek model successfully created a functioning FakeRandom dice, as seen below.

```java
/**
 * Returns the next value from the sequence, adjusted for die sides
 * @return Roll result (1 to sides)
 * @throws IllegalStateException if no more values available
 */
@Override
public int roll() {
    if (!valueIterator.hasNext()) {
        throw new IllegalStateException("No more values in sequence");
    }

    int nextVal = valueIterator.next();
    // Convert value to 1..sides range
    return ((nextVal - 1) % getSides()) + 1;
}
```

The last part of this program was to clone the FifteenPlayer class and create a SevenPlayer class, and then use JUnit to perform unit testing on both the SevenPlayer and FakeRandom class. I had to modify my SevenPlayer class to override the rollDie method so that it could use a different die. Then I used my IDE to generate the FakeRandomTest class with JUnit, and then I prompted the AI:

Create unit testing for both my FakeRandom and SevenPlayer class.

This was also followed by the code for both classes.

The DeepSeek model took 5 minutes to created 3 simple unit tests to verify that the FakeRandom class functioned properly. It also created five unit tests for the SevenPlayer class, which looked a lot like the unit tests we did in class to verify that the player functions properly in many circumstances, such as rolling a 6, rolling over a 6, and rolling the correct accumulated score. Overall, I couldn't find any other tests to implement that weren't redundant, and executing the unit tests successfully passed all 8 tests. Below are some useful snippets of the unit tests that passed, similar to those created in class.

```
class FakeRandomTest {

    // ======================
    // SevenPlayer Tests
    // ======================

    @Test
    void sevenPlayer_returnsZeroWhenSixRolled() {
        List<Integer> fixedValues = Arrays.asList(1, 6); // Roll a 1, then a 6
        FakeRandom fakeDice = new FakeRandom(6, fixedValues.iterator());
        SevenPlayer player = new SevenPlayer();
        player.setDice(fakeDice);

        int score = player.play();
        assertEquals(0, score, "Score should be 0 when a 6 is rolled");
    }

    @Test
    void sevenPlayer_stopsAtSevenOrMore() {
        List<Integer> fixedValues = Arrays.asList(4, 3); // 4 + 3 = 7
        FakeRandom fakeDice = new FakeRandom(6, fixedValues.iterator());
        SevenPlayer player = new SevenPlayer();
        player.setDice(fakeDice);

        int score = player.play();
        assertEquals(7, score, "Score should be 7 when target is reached");
    }

    @Test
    void sevenPlayer_accumulatesUntilTarget() {
        List<Integer> fixedValues = Arrays.asList(2, 3, 4); // 2 + 3 + 4 = 9
        FakeRandom fakeDice = new FakeRandom(6, fixedValues.iterator());
        SevenPlayer player = new SevenPlayer();
        player.setDice(fakeDice);

        int score = player.play();
        assertEquals(9, score, "Score should accumulate to 9, exceeding target");
    }

}
```