Bulldog: Strategy Pattern

Author:
Michael Yattaw

April 16, 2025

The bulldog application has had many improvements and code changes through each part of the program, so again we are modifying the code to implement a Strategy desgin pattern architecture. The Strategy design pattern is a behavioral design pattern that was implemented to encapsulate the game status, which can allow different player classes to use their own algorithms to decide when to stop or continue playing for a round.

To begin implementation, I reviewed all my player classes to identify potentially duplicated code in the play method. With a better understanding of how the GameStatus class and continue method might be structured, I started prompting the DeepSeek AI model, which look like this:

> Create a class to encapsulate the status of the game. This should be a relatively simple class and should refactor my Player class play method. This refactoring also involved creating a new method that provided each Player's decision about whether or not to continue rolling in this round.

Then the prompt was followed by the code for my player class. Below, you can see the newly updated methods produced by the AI model. It took roughly 5 minutes to create the prompt, have the AI model process it, and generate the code.

```java
public int play() {
    GameStatus gameStatus = new GameStatus();

    while (!gameStatus.isTurnOver()) {
        int roll = rollDie();
        gameStatus.setLastRoll(roll);

        if (roll == 6) {
            gameStatus.setRoundScore(0);
            gameStatus.setTurnOver(true);
        } else {
            gameStatus.setRoundScore(gameStatus.getRoundScore() + roll);
            if (!continueRolling(gameStatus)) {
                gameStatus.setTurnOver(true);
            }
        }
    }

    this.score += gameStatus.getRoundScore();
    return gameStatus.getRoundScore();
}

public abstract boolean continueRolling(GameStatus gameStatus);
```

As seen in the code above, each play creates a new object to track round scores, a Boolean indicating whether the round is over, and the value of the last roll. Consequently, player classes should now focus solely on determining actions after a roll, rather than handling redundant game logic, such as rolling a 6.

Now that I had a properly structured player class, I needed to update all my other player classes to use my new continueRolling method. Initially, I tried to manually refactor my code, but noticed that copying the code from the play method to the continueRolling method would take a while. So, in the same chat with the original prompt, I began prompting the AI for each player class with prompts that looked like this:

> Refactor this player subclass to match my new player:

Then I pasted the entire player subclass code that followed. For each class, it took close to 3 minutes for the Deepseek AI model to process, and all I needed to do was copy the generated continueRolling code into my player class and delete the old play method. As shown below, the implementation of the continueRolling method was much clearer than the old, redundant play methods, which reused the same code.

```java
/**
 * Determines whether the human player wants to continue rolling.
 * Prompts the user for input after each roll (unless a six was rolled).
 *
 * @param gameStatus The current game status information
 * @return true if player wants to continue rolling, false otherwise
 */
@Override
public boolean continueRolling(GameStatus gameStatus) {
    if (gameStatus.getLastRoll() == 6) {
        System.out.printf("   Player %s rolled 6, so player scored 0 for the turn.%n", getName());
        return false;
    }

    System.out.printf("   Player %s rolled %d, cumulative score is %d%n",
            getName(), gameStatus.getLastRoll(), gameStatus.getRoundScore());

    System.out.println("Type 'stop' to end turn or press Enter to continue:");
    String input = SCANNER.nextLine().trim();

    if (input.equalsIgnoreCase("stop")) {
        System.out.printf("   Player %s chose not to continue, scoring %d for the turn%n",
                getName(), gameStatus.getRoundScore());
        return false;
    }

    return true;
}
```

The DeepSeek AI model saved me a lot of time by automatically updating the Javadoc comments for the new continueRolling method, while also transferring the old logic from the play method. Although I spent a little over 20 minutes prompting the AI for all the classes, DeepSeek ensured consistency and provided fully functional code. As a result, the Deepseek model performed well, leaving very minimal coding tasks for this part of the program.