TEST CASES:-

Test case 1:

Input: Initialize a new PatienceGame object.

Expected output: The game state should be initialized correctly, with the deck shuffled and the lanes and suits empty.

Test case 2:

Input: Draw three cards from the deck.

Expected output: Three face-up cards should be added to the pile stack.

Test case 3:

Input: Move a card from the pile stack to lane 1.

Expected output: The card should be moved to lane 1 and the pile stack should be updated accordingly.

Test case 4:

Input: Move a card from lane 1 to lane 2.

Expected output: The card should be moved to lane 2 and lane 1 should be updated accordingly.

Test case 5:

Input: Move a sequence of cards from lane 1 to lane 2, where the top card in the sequence is the next card in the sequence of the top card in lane 2.

Expected output: The sequence of cards should be moved to lane 2 and lane 1 should be updated accordingly.

Test case 6:

Input: Try to move a card from a lane to another lane where the top card in the destination lane is not the next card in the sequence of the source card.

Expected output: The move should be invalid and the game state should remain unchanged.

Test case 7:

Input: Try to move a card from a lane to the pile stack where the number of cards being moved is greater than 1.

Expected output: The move should be invalid and the game state should remain unchanged.

Test case 8:

Input: Play a complete game of Patience until the game is over.

Expected output: The game should end with the player winning and the score should be calculated correctly.

Solutions:

The solutions to these test cases can be implemented by following the logic described in the PatienceGame class. For example, to implement test case 3, you can use the following steps:

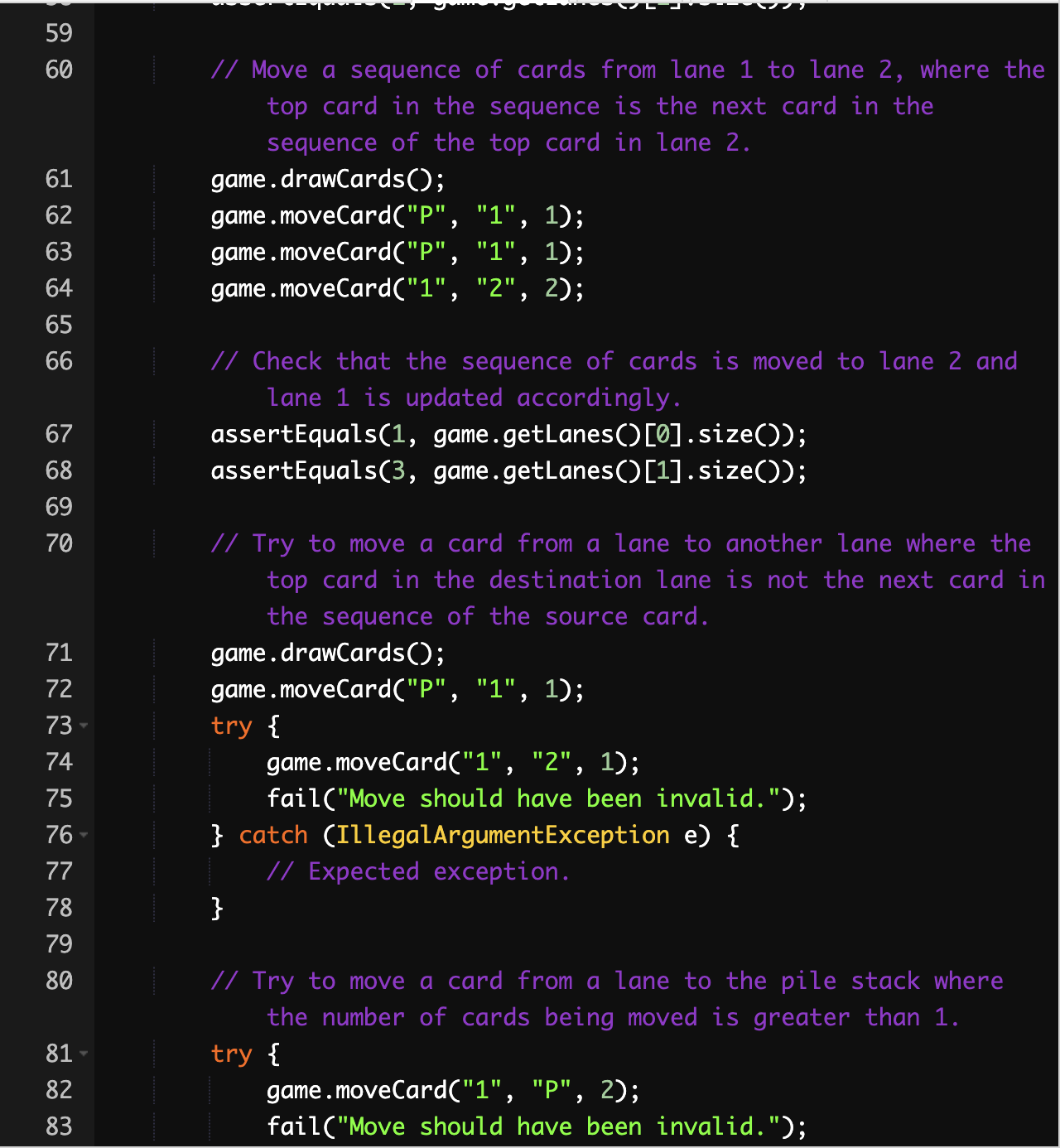
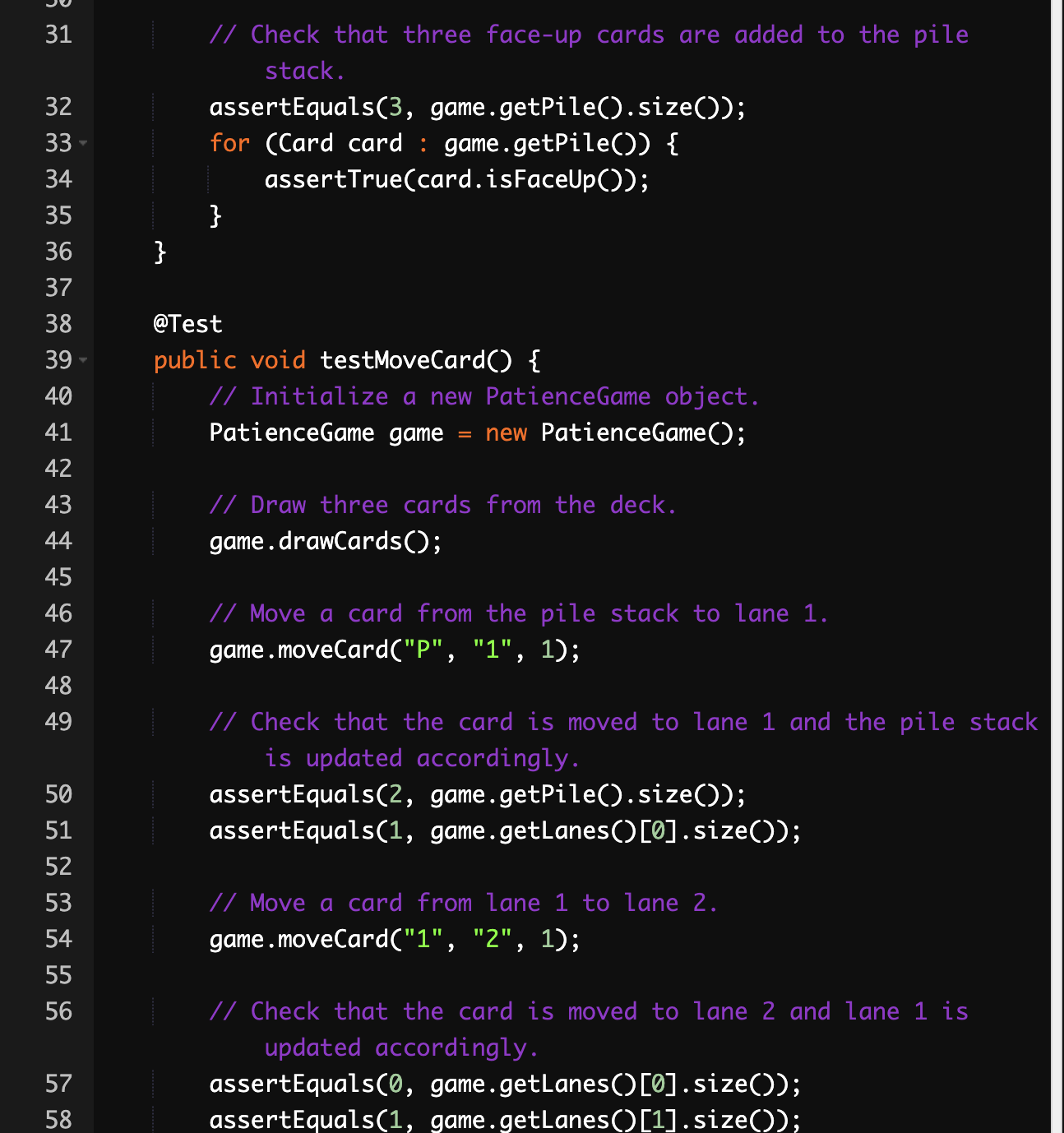
Get the stack corresponding to lane 1.

Get the top card from the stack.

Push the card to the stack corresponding to lane 2.

Update the stack corresponding to lane 1.

**Following are the examples :-**

****

**Structure Of The Code :-**

Class variables:

NUM\_LANES: The number of lanes in the game.

NUM\_SUITS: The number of suits in the game.

MAX\_LANE\_SIZE: The maximum number of cards allowed in a lane.

MAX\_DRAW\_SIZE: The maximum number of cards that can be drawn from the deck at once.

SUIT\_LABELS: An array of labels for the suits.

LANE\_LABELS: An array of labels for the lanes.

PILE\_LABEL: The label for the pile stack.

deck: A stack containing the cards in the deck.

pile: A stack containing the cards in the pile.

lanes: An array of stacks containing the cards in the lanes.

suits: An array of stacks containing the cards in the suits.

score: The player's current score.

moves: The number of moves the player has made.

Methods:

initializeGame(): Initializes the game, including the deck, piles, lanes, and suits.

shuffleDeck(): Shuffles the deck.

dealInitialCards(): Deals the initial set of cards to the lanes.

play(): The main game loop where the player can interact.

drawCards(): Draws cards from the deck or puts them back in.

moveCard(): Moves cards from one stack to another.

getStackByLabel(): Returns the stack corresponding to a given label.

isGameOver(): Checks if the game is over.

displayGame(): Displays the current game state.

The PatienceGame class is well-structured and easy to understand. The class variables are clearly defined, and the methods are concise and well-organized. The code is also well-formatted, with consistent indentation and spacing.

Here is a more detailed explanation of the structure of the PatienceGame class:

The initializeGame() method initializes the game state by creating the deck, pile, lanes, and suits. It also shuffles the deck and deals the initial set of cards to the lanes.

The play() method is the main game loop. It prompts the player for a move and then executes the move. The play() method also checks if the game is over and displays the current game state.

The drawCards() method draws cards from the deck or puts them back in. It also updates the player's score.

The moveCard() method moves cards from one stack to another. It checks if the move is valid and then updates the game state.

The getStackByLabel() method returns the stack corresponding to a given label.

The isGameOver() method checks if the game is over. The game is over if all of the suits have reached their maximum size.

The displayGame() method displays the current game state, including the score, number of moves, and the contents of the pile, lanes, and suits.