**Name:** Conor Carney

**Student Number:** 20252987

**Course:** HDIP Software Development

**Overview of Game**

The game I have chosen to make is a computerised version of the classic Irish card game “25”, played via the command line. This game uses a standard deck of 52 cards, no jokers, and is played with between 2 and 10 players. In this instance there is one human player, with the number of computer players chosen by the player. The objective of the game is to win 5 ‘tricks’, each worth 5 points, adding up to a total score of 25.

The rules of the game are as follows: 5 cards are dealt to each player, with the card remaining on the top of the deck turned up after dealing is done. This is the trump card.

Numbered reds are high, while numbered blacks are low- E.g. A red 10 will beat a red 7, but a black 7 will beat a black 10.

The picture cards (Jack, Queen, King, Ace) are valued in order, and will beat numbered cards. E.g. A King will beat a Queen or a Jack but lose to an Ace of the same suit.

The first player to play each round decides the suit of the round, and players must follow suit. E.g. If the first card played that round is a club, the following players must play a club (if they have one).

However, a trump card can be played in place of following suit, and any trump card wins over a card following suit. However this trump card can be beaten by a higher valued trump.

In the case of trumps, the 5 is the most valuable card, it will beat any card played, the Jack of trumps is the second most valued card, then the Ace of trumps. The Ace of Hearts is always the fourth best card, unless it is the ace of trumps, then it is the third best card.

The player who wins starts the next round. If all 5 cards are played the scores remain the same and a new hand is dealt, until one player reaches 25.

**How to Play**

A copy of the rules will be printed immediately to the screen, and then you will be prompted to enter the number a players you would like in the game in the command line, between 2 and 9. An input outside this will result in an error message and you will be asked to enter a number again. Then the cards are dealt to each player and the game begins.

The cards in your hand are numbered, and you enter the number to the corresponding card to play. Entering a number not visible will result in an error message and you will be asked to enter a number again.

The computer players are programmed to play their best cards from the start, depending on the starting suit and trump. At the end of each round, the scores for all players are given, and your remaining hand is shown, and you will be asked to enter your next card.

Once all 5 cards are played, a new hand is dealt to each player and play continues until one player reaches a score of 25.

Once a player reaches 25 they are announced as the winner and the user is prompted to play again.

**Code Structure**

The code is structured in a single java class called Game. java.util.Scanner, java.util.ArrayList and java.util.InputMismatchException are imported. The code is structured as follows: Class variables are initialised, then the main method is called. The Deck method follows, then in order: the Deal method, SetTrumpValues method, PlayCard method, CheckHands method, CheckWinner method, ResetCurrentRound method, CheckScore method and finally the constructor, getter, and setter methods for constructing player and card objects.

Class variables are initialised, before the main method begins. Then rules are printed to screen, and the rest of the main method is contained inside a do while loop- this controls whether the person would like to play again at the end of the game. The initial input for number of players is taken from the user, error checked to ensure it’s an integer value between 2 and 9, and an ArrayList of Game objects is created, with each containing a player’s number of cards, score, and current hand.

Then the Deck method is then called from the main method. The Deck method takes no inputs and returns an ArrayList<Game>. It uses for loops to create a 52 card deck of individual cards, that were Game objects. I choose to use the Game objects over simply using Strings as it was easier to compare/change values depending on the colour/suit etc. The deck is then shuffled using a small algorithm and Math.random.

Then, a second do while loop is use, which holds the methods for a single game of 25, which loops until one player has reached a score of 25. The Deal method is called, which saves each card dealt in the individual players object as currentHand, and then removes the card from the deck using a remove method. After all the players are dealt their hand, the card at [0] in the deck is shown, which is saved in a Game variable as the trump card.

Next, the SetTrumpValues method is called. This sets the values of the 5 and Jack of trump to the best and second best value in the game, respectively, by checking each players hand to see if they contain either using a for loop. It was done like this as the trump suit change with each deal – so it’s only valid for the current deal.

The currentRound variable is then called, which will contain the cards played from each player in order, stored in an ArrayList<Game>.

A while loop is used until all cards in this deal are played, controlled by a numOfCards variable. Inside this while loop is the CheckHands, Play Cards, Check Winner and Check Score methods. The CheckHands method uses a for loop to print out the players current hand. The PlayCards method checks to see the previous round winner, as they start the next round, using a for loop. Nested for loops the control the playing of each card – the human player’s input is error checked for an integer value of the cards listed in their hands. The computer players play cards based on an algorithm of if else statements- they will play a trump if possible, otherwise follow suit, playing their highest possible card. If they have no trump or suit, they will play their first card.

The numOfCards variable is then reduced by 1.

Next, the CheckWinner method is called, using nested for loops and if else statements to see which players card had the highest value. It is saved in a currentWinner variable, which is used to add the score to the player object who won, and change their started variable to true so they start the next round. The round winner is printed off. The currentRound variable is then reset to an empty ArrayList.

The CheckScore method prints off the current score of each player and checks to see if a player reached 25. If they have, the isWinner variable is returned true, and the while loop in the main method is exited. If not isWinner is returned false.

Finally, after the game has had a winner, the user is prompted to play again, and this is also error checked. Should the user decide to play again the do while loop continues, otherwise the program exits.

**Data Structures and Algorithms Used**

Data Structures – Arrays and ArrayLists.

Arrays were used as the amount of data is small – 52 cards and 5 card hands, insertion and deleting loops, and no sorting was necessary. I used ArrayLists for the deck and hands, due to their dynamic sizing and easily accessible methods. Although it is slightly slower than using normal arrays, in this project, memory wise the numbers are insignificant, and the ease of use in editing the various ArrayLists made it an obvious choice. The size of an ArrayList is also easily modified, and the size controls several of the functions of the project. An ArrayList is roughly similar to a vector data structure, but unsychronised (which wasn’t an issue here). As we were not using primitive types and we didn’t need to use wrapper classes, and I felt an ArrayList was the best choice.

Algorithms

Simple if else statements inside nested for loops were used for the algorithms in this project. The main algorithms were used to initialise the deck and shuffle in the Deck method, to programme the computer to play a card according to the game rules in the PlayCard method, and in the CheckWinner method to see who won each round. Wrapping around the ArrayList so the previous round winner started first in the PlayCard method was probably the most complex algorithm, in part because it was added late in production, meaning it had to be worked around the original code. This changed the conditions of the for loops in the PlayCard method and the CheckWinner methods, as they were all based around the array staying in order. The shuffle function in the Deck method was also a nice little algorithm, setting the deck using temporary variables and math.Random.

**Learning Outcomes**

Building this project definitely grew my knowledge of ArrayLists various functions, and I found several aspects challenging in this regard. It took a while to get used to using the different call methods for them, and accessing them was difficult at points due to the structure of the objects I was using (An ArrayList of player objects each containing an ArrayList of card objects each with a suit/value/colour), which showed the value of having a well thought out design. Understanding the structure was the key to making the project work.

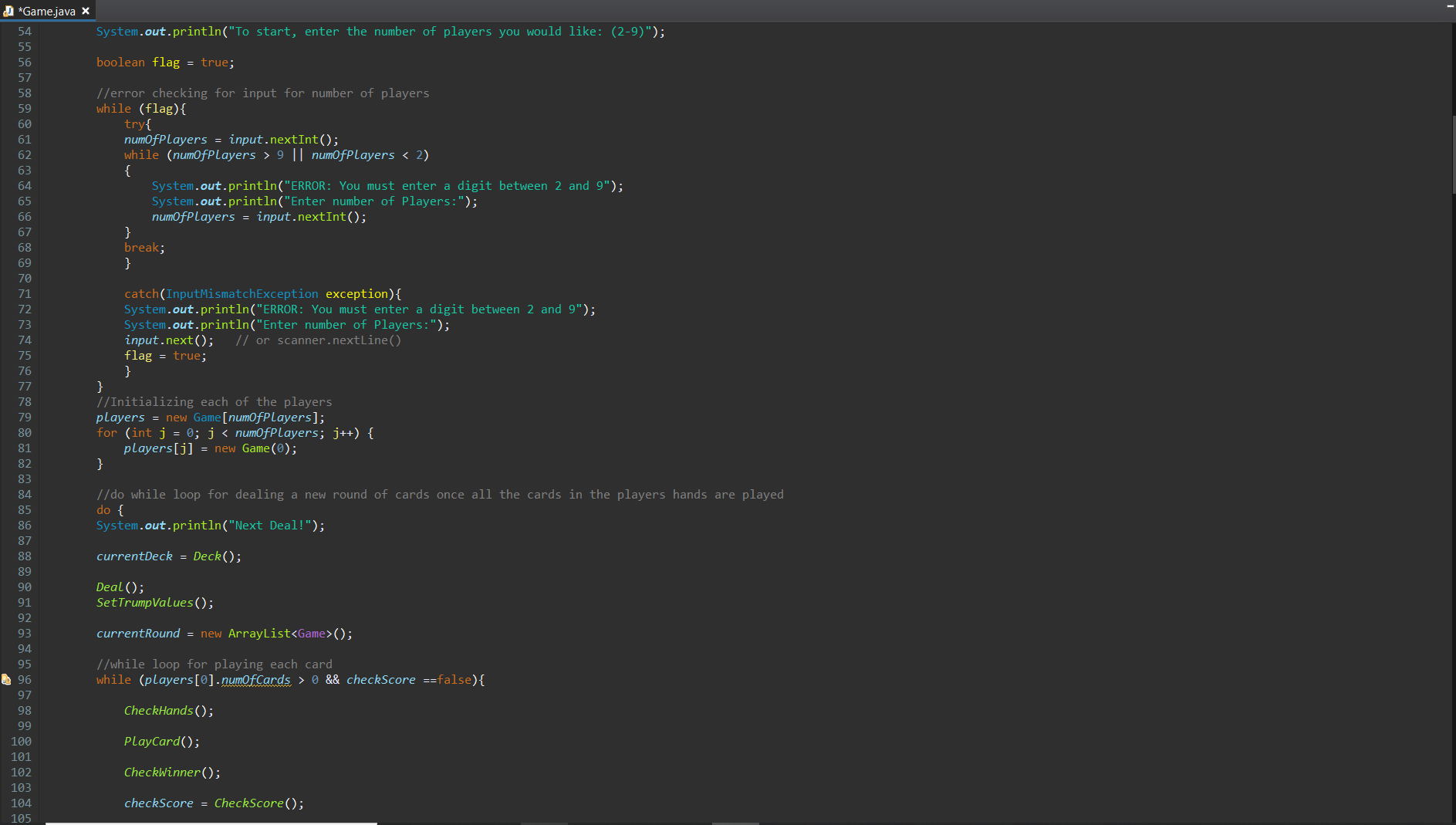
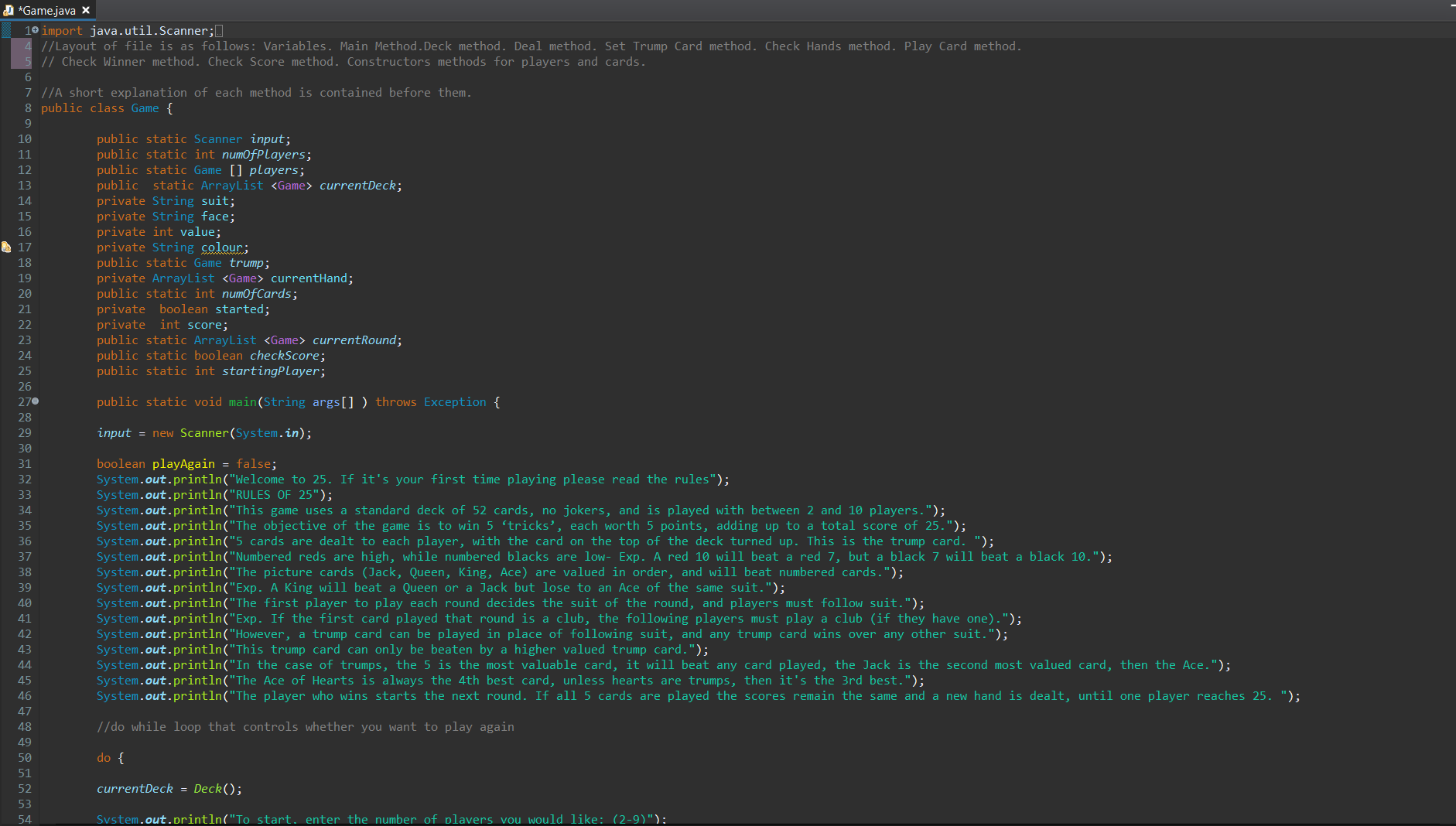
Using a variety of different protections and methods cemented the difference between static/non static, private/public methods properly in my head.

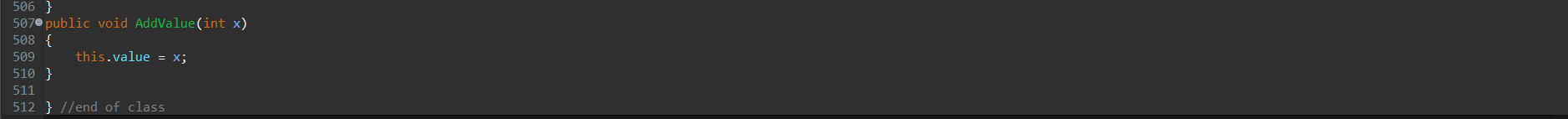
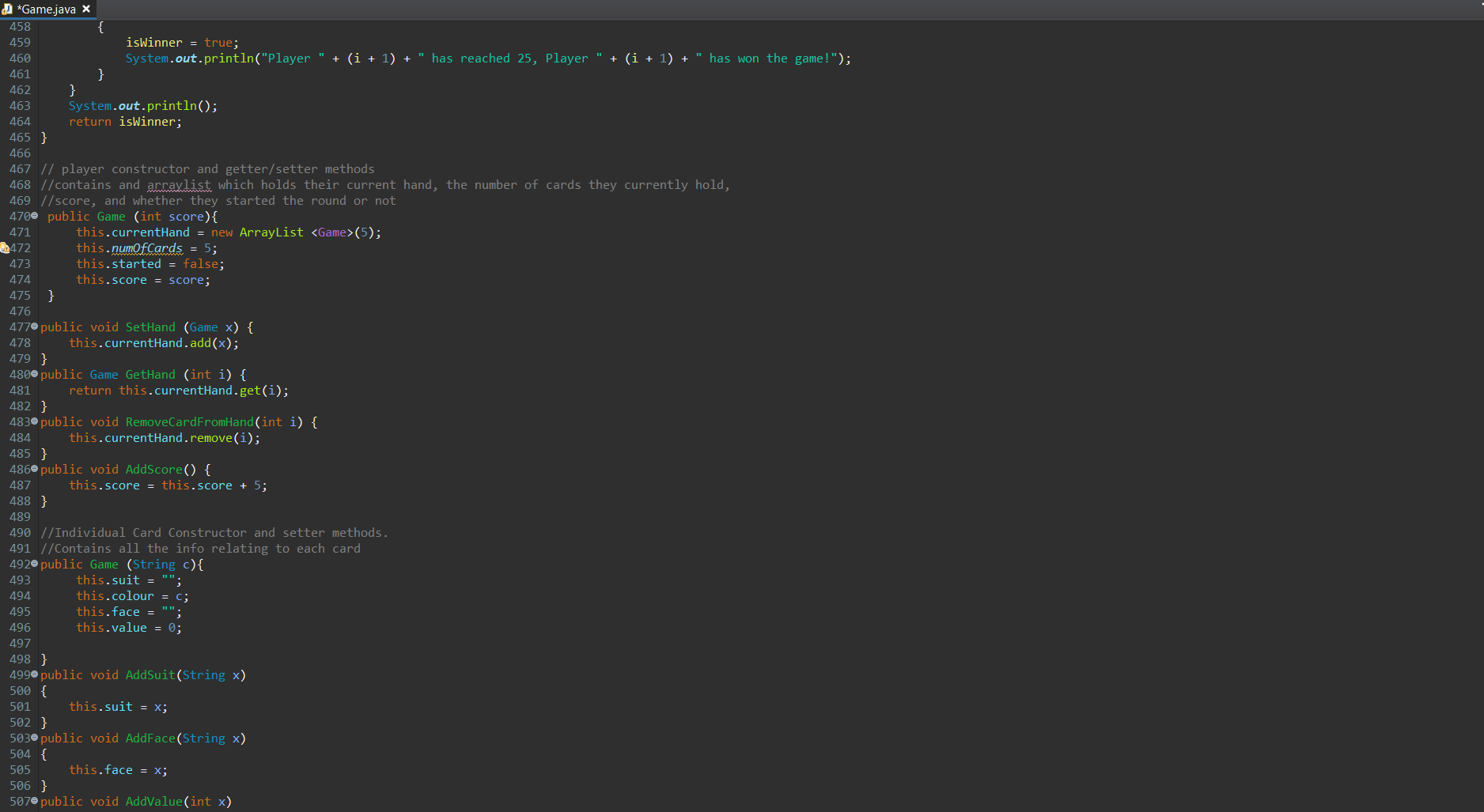
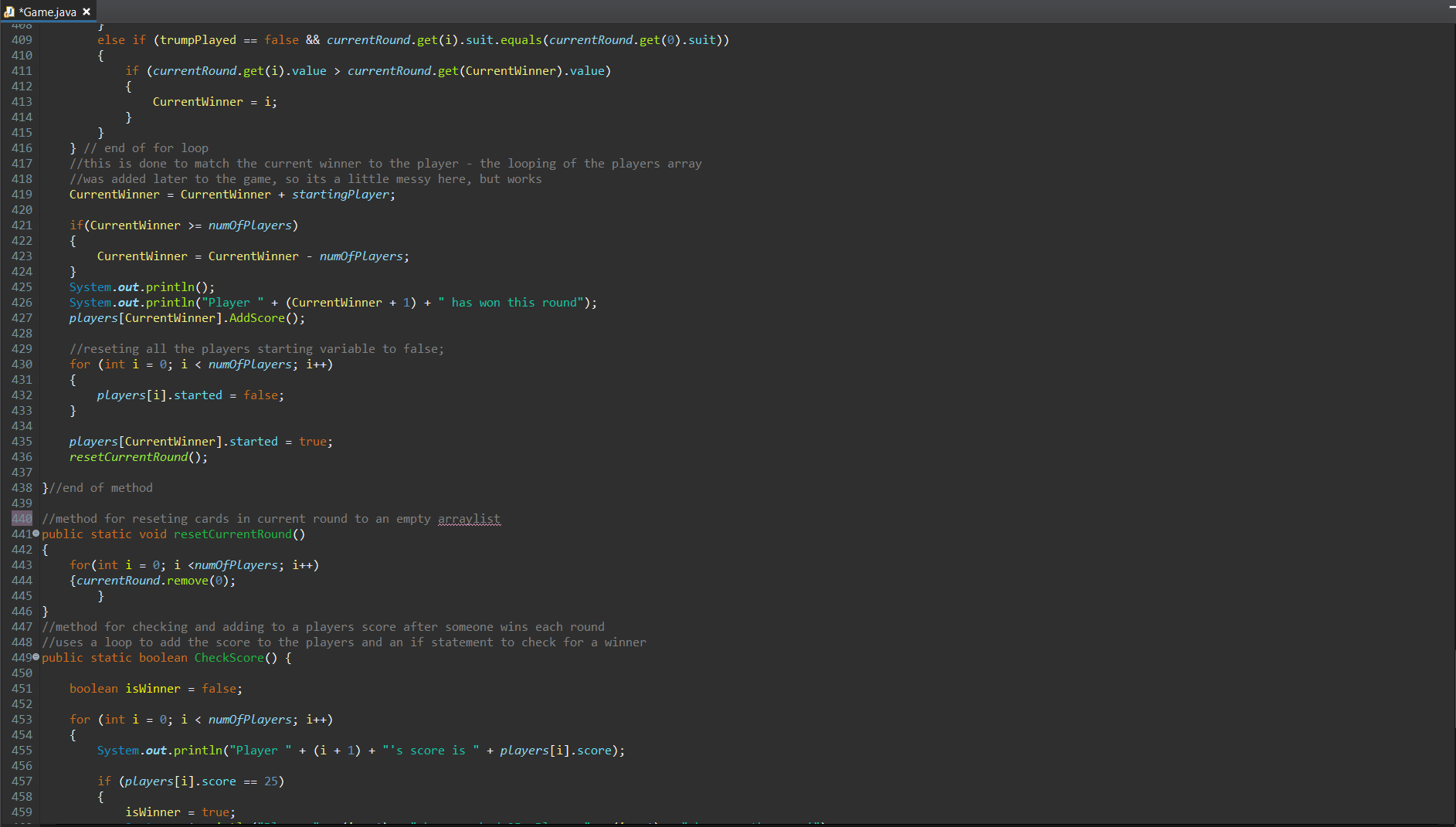
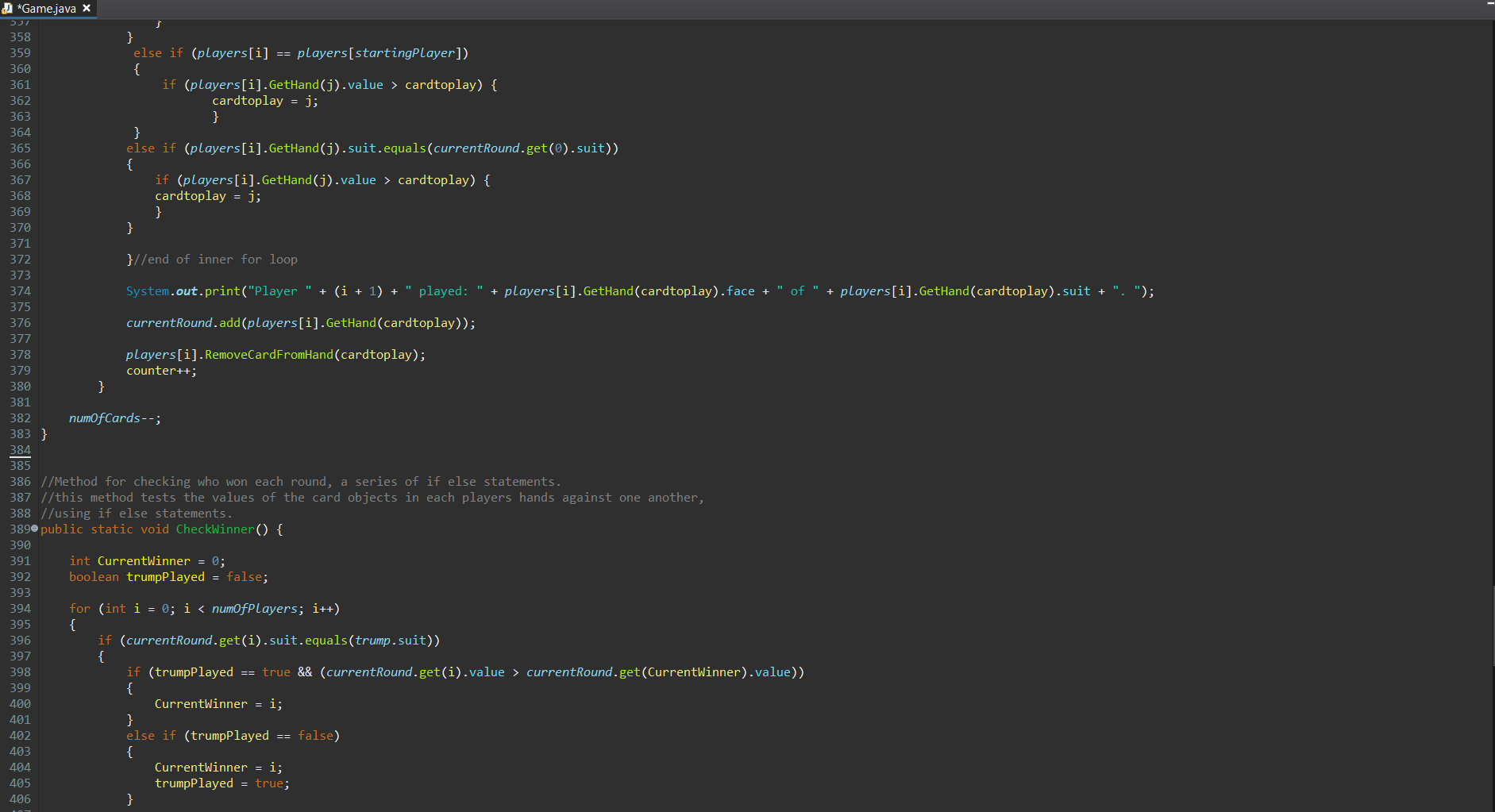
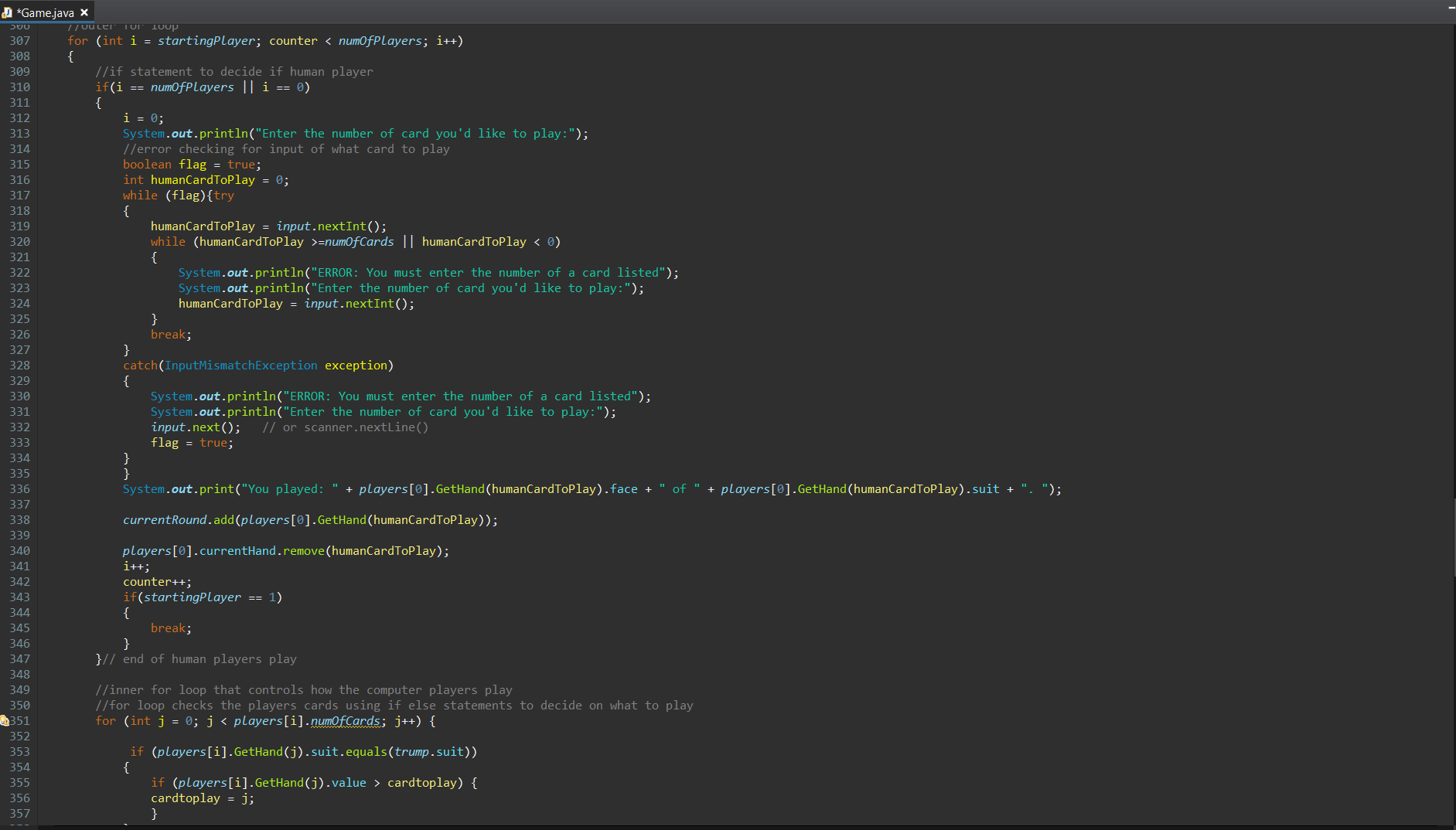
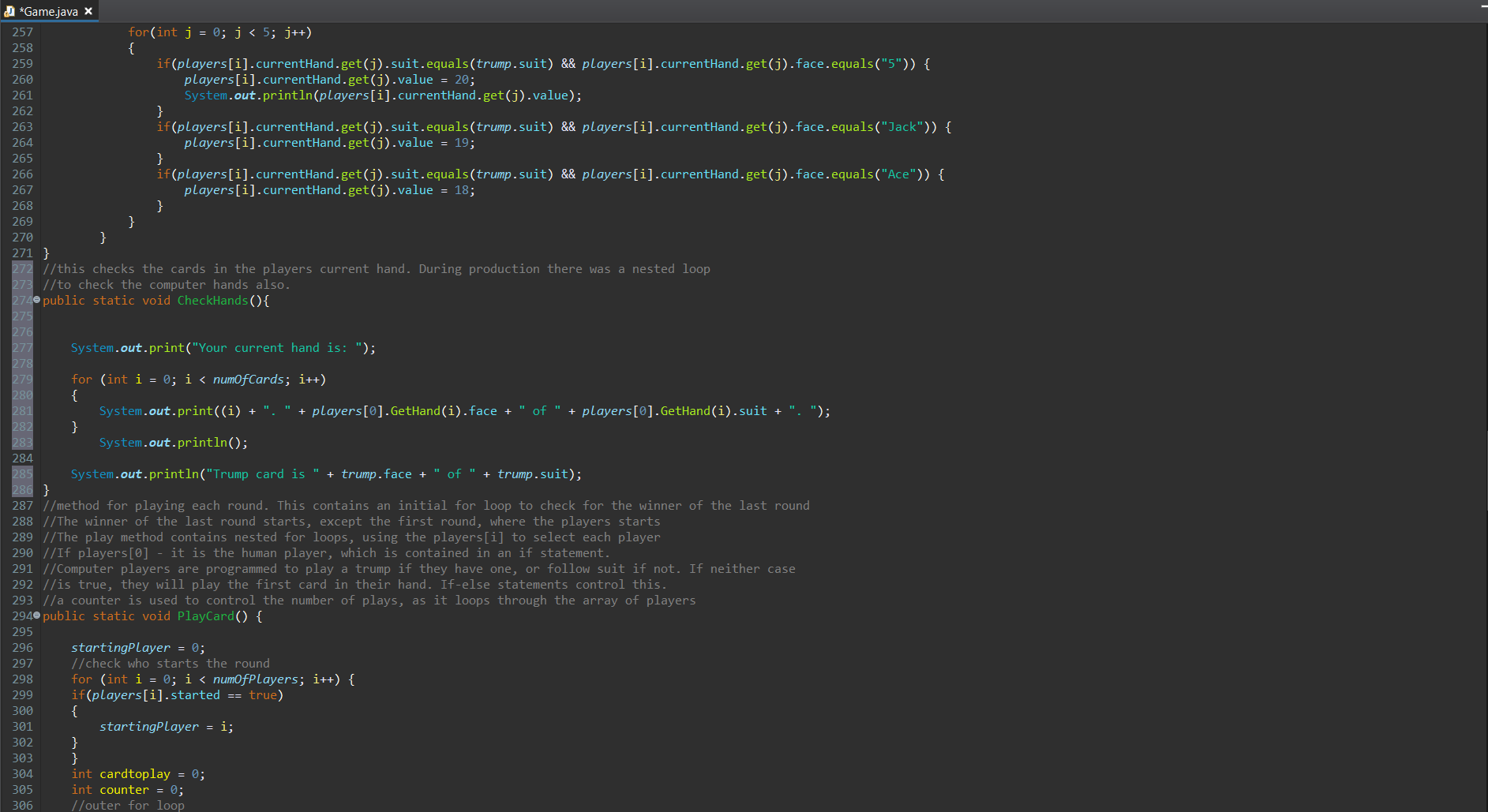
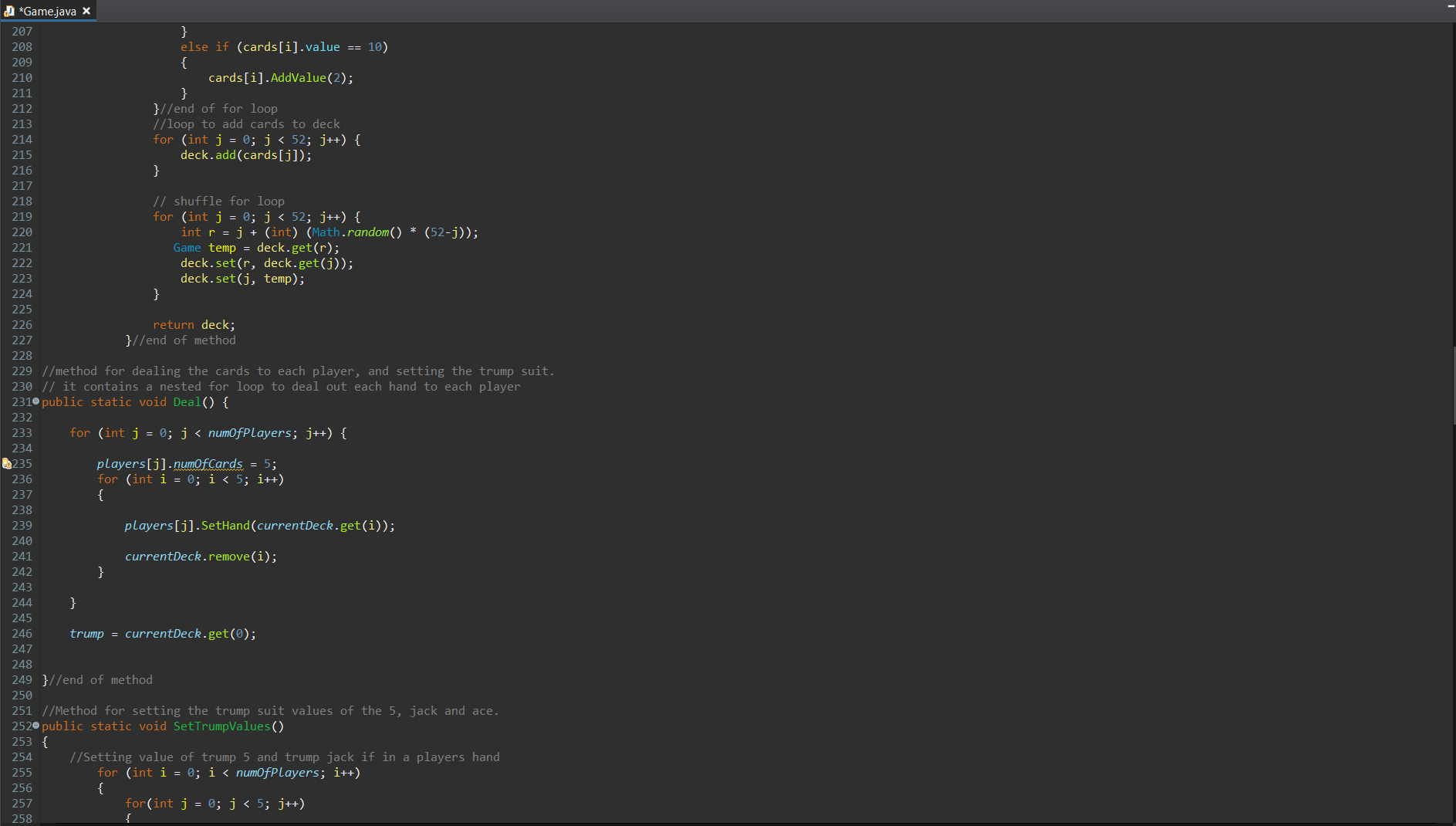
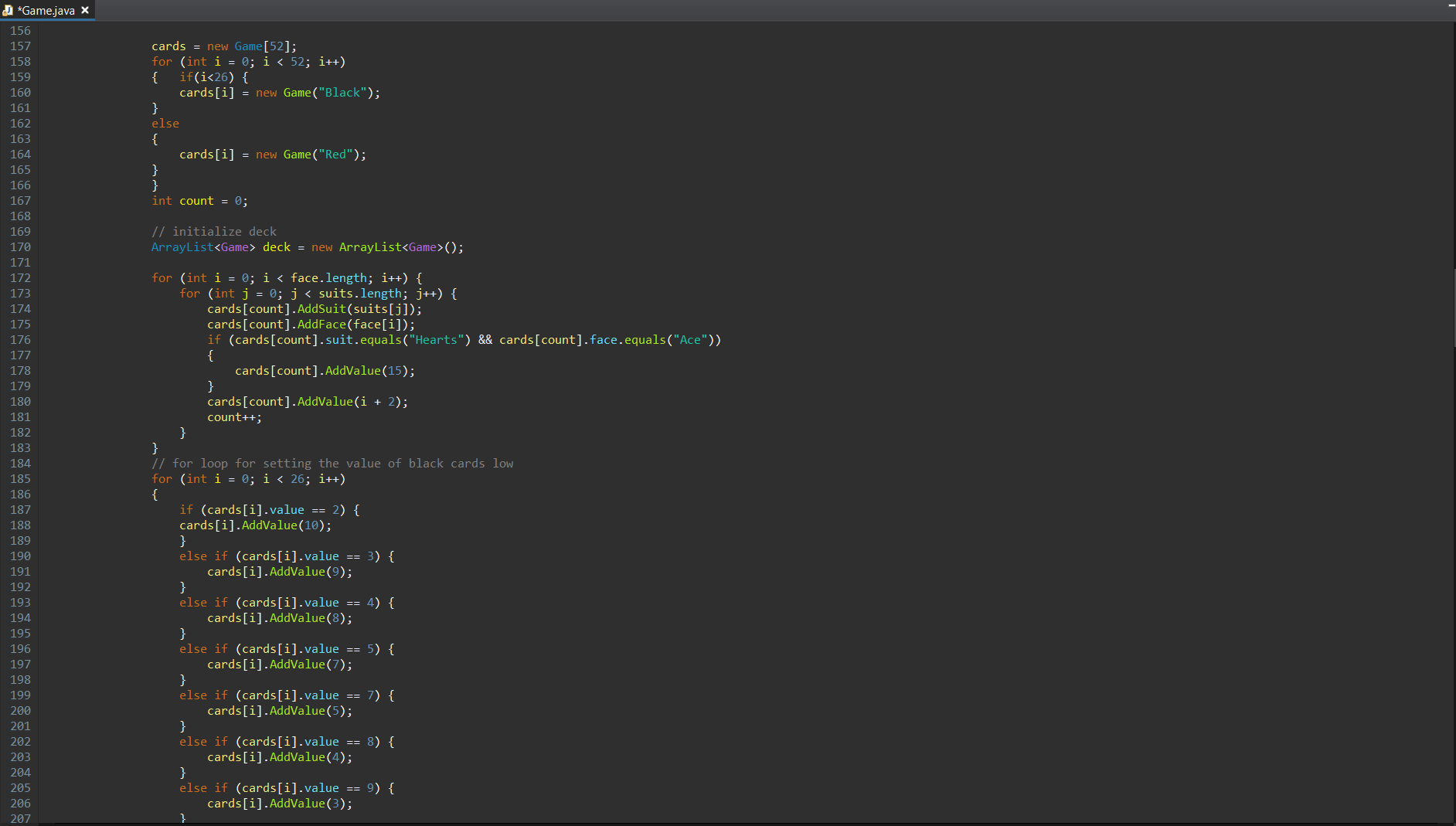
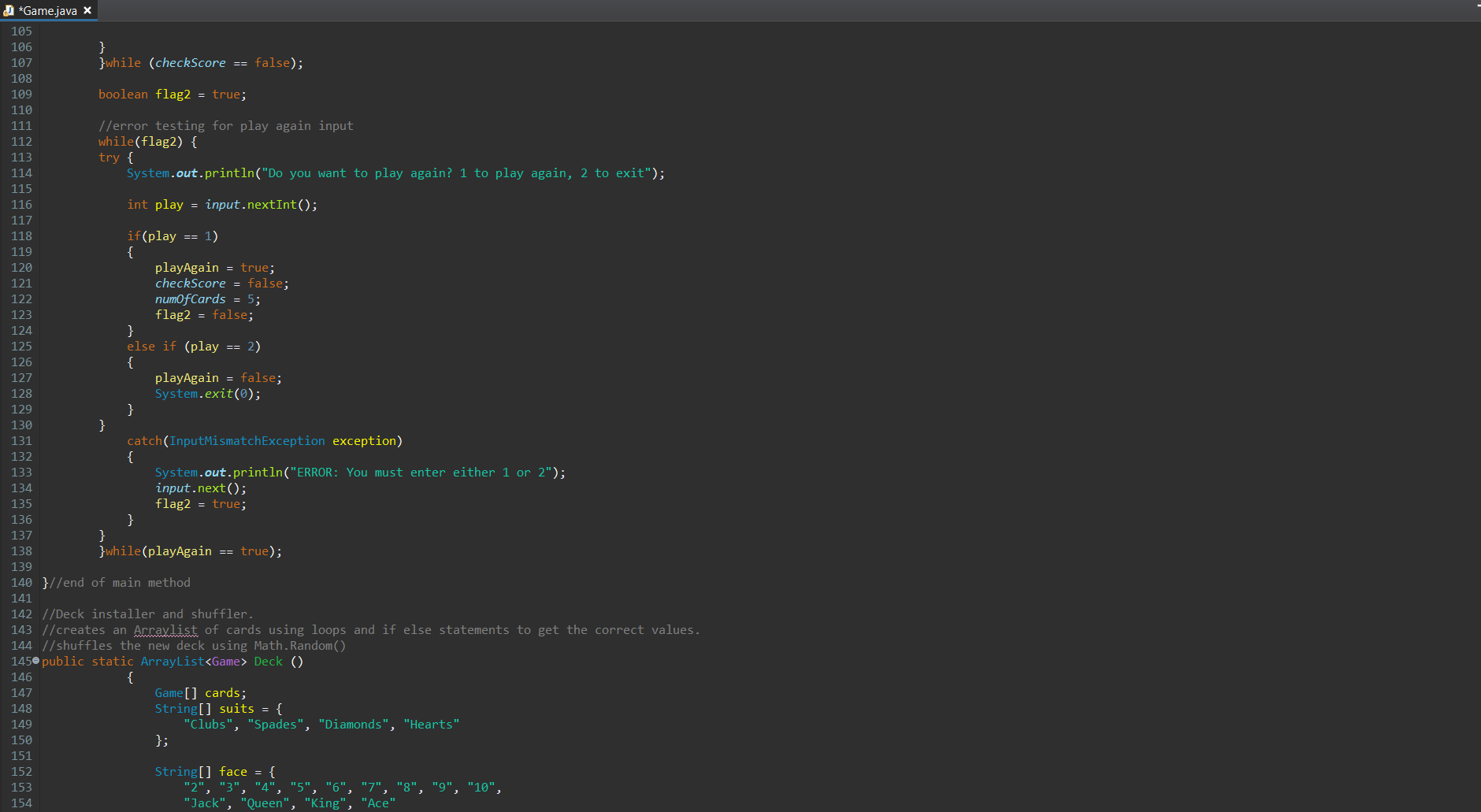
Adding in error testing of various inputs was useful, I hadn’t much experience with try catch blocks and it took a little bit of playing around to get used to them.

I appreciate the camaraderie of working in a team to solve problems much more now – There were times where I was stuck on a problem and it caused a bottleneck in the project for several days, and was checking the wrong part of the code for the solution, when a simpler solution was right in front of me. A team would have been much more time efficient as an outside perspective would have noticed this much quicker. Also having others to bounce ideas off would be nice.

Future additions to this project could include adding graphics, a la Microsoft Hearts, so the player can see their own hand and the other cards been played. Adding an option for additional human players is an additional possibility, as is having different difficulty levels algorithms for the computer players. Another aspect to make the game more interesting could be adding in a pot for betting, and saving the players score/money for future games. Future projects could also move online, as games such as online poker have exploded in popularity in the previous years.

**Appendices**

****

****