Roshan Patel

CS111.BlackJack Documentation

Rules:

My Blackjack game asks the user for game preferences and sets up a game between players and the dealer. The dealer is not a player; he/she is the house that the players are playing and betting against.

Players may ask for hints, hit/stand, double down, and split when applicable. All inputs for game actions are Boolean responses from the user. Bets and initial Bank values are exceptions as double inputs.

The game can be continuously played until the Dealer loses or the user wants to quit. The dealer can be changed between rounds.

Process:

My process in designing the Blackjack game was to construct key objects and have them all interact within a main Blackjack.java file. The key objects I made were the Card, Deck, Hand, and Player.

Card being the most basic, consists of a value and face. A deck is a single array of Card objects. A Hand is a malleable array of Card objects. Finally, a Player was a money holding hand who could be dealt cards, hit cards, etc.

I started by very basically defining each object in relation to each other with the help of the hints posted on Sakai. I would test each individual method in a temporary main file called workspace.java. Once I finished constructing a deck of cards, shuffling, and dealing them out to multiple players, I started working on my main Blackjack.java file.

I looked to Sakai for help on starting my Blackjack main method. I decided it would be easiest to construct a sub-method in Blackjack.java that would run though each player in an array and have them run through rounds (methods called from Player.java). Organization was key to my steady completion of this project. Had I not properly commented and indented my code, it would be an illogical mess.

After error checks and debugging and I was satisfied with my basic Blackjack game so far, I decided to implement a money system. It made the most sense to me to create bets and a bank as an attribute of the Player class. After some more error checks, I was able to implement a wagering system into my game.

Now I decided I wanted to give the users the option to double down and split their hand if possible. Double downing, which is doubling your bet for only 1 more hit, was very easy to add to my code. Splitting, however, caused me a lot of trouble. I realized that by splitting a Player class’s hand, there would be two hand objects that belonged to a Player that were both going through the Player methods of playing rounds and starting rounds. I had to restructure a lot of my code in the playing rounds methods in order to account for the possibility of a Player having two hands and two separate dynamic bets. As time consuming and tedious as it was, I was finally able to successfully implement splitting into my game. Too bad the possibility of being able to split is relatively low.

Finally, I added a hint system as a sub-method in my Blackjack.java code. It is a static probability that reports a percentage chance the user will bust if he/she hits the next round. Hints are offered at the beginning of each round for each player.

Efficiency Analysis:

The efficiency of my Blackjack game is primarily dependent on the deck algorithms and play round algorithms since these are the most referenced

The deck constructing algorithm uses 3 nested for loops. The outer loop runs the number of decks the user inputs. The middle loop runs for 4 suits. The inner loop runs for 13 face types. The efficiency of this algorithm if the number of decks is N is N^3^4 = N^12. This results in an exponential BigOh function [ O(n^12) ]. This means that the algorithm is not very efficient for large values of N.

The deck shuffling algorithm is a single for loop that runs for the length of the deck. It runs through each element of the deck and swaps it with a random location within the deck. Therefore the BigOh notation of this algorithm is just N, the number of elements in the deck. This is a linear BigOh notation.

All of the play round algorithms are located within the Player object as methods. The Blackjack,java code calls these methods in a for loop that run the length of the number of players there are. This means that the algorithm’s BigOh notation is a linear function dependent on the number of players playing the round.