CS231 Project 1: Blackjack

September 11, 2023

Abstract

This project is meant to teach CS Students how to build classes and manipulate Arraylists in Java. This is done through simulating a simple version of the card game Blackjack, involving many objects such as a Card, a Deck, a Hand, and Game class calling and manipulating one another to eventually be able to display the result of each gameplay on the terminal.

Results

The BlackjackTests program requires better tests to verify that each method was functioning correctly, I decided to create tests that checked playerTurn(), dealerTurn(), reset(), and deal():

```java
public class BlackjackTests {

    public static void blackjackTests() {

        // case 1: testing Blackjack() and Blackjack(i)
        {
            // set up
            Blackjack bj1 = new Blackjack();
            Blackjack bj2 = new Blackjack(reshuffleCutoff:10); // shuffle

            // verify
            assert bj1 != null : "Error in Blackjack::Blackjack()";
            assert bj2 != null : "Error in Blackjack::Blackjack()";
            bj1.game(verbose:true);
            boolean playerResult = bj1.playerTurn();
            // testing playerTurn()
            if (bj1.Player.getTotalValue() > 21) {
                assert playerResult == false;
                System.out.println(playerResult + " == false");
            } else {
                assert playerResult == true;
                System.out.println(playerResult + " == true");
            }
            System.out.println(x:"*** Player Turn Works! ***\n");

            // testing dealerTurn()
            boolean dealerResult = bj1.dealerTurn();
            if (bj1.Dealer.getTotalValue() > 21) {
                assert dealerResult == false;
                System.out.println(dealerResult + " == false");
            } else {
                assert dealerResult == true;
                System.out.println(dealerResult + " == true");
            }
            System.out.println(x:"*** Dealer Turn Works! ***\n");

            // testing reset()
            bj2.reset();
            assert bj2.Player.Hand == null;
            assert bj2.Dealer.Hand == null;
            System.out.println(bj2.Player.Hand + " == []");
            System.out.println(bj2.Dealer.Hand + " == []");
            System.out.println(x:"*** Reset Works! ***\n");

            // testing deal()
            // System.out.println(bj2.Deck);
            bj2.deal();

            assert bj2.Player.Hand.size() == 2;
            assert bj2.Dealer.Hand.size() == 2;
            System.out.println(bj2.Player.Hand.size() + " == 2");
            System.out.println(bj2.Dealer.Hand.size() + " == 2");

            System.out.println(x:"*** Deal Works! ***\n");

            // testing the shufflecutoff()
            while (bj2.Deck.size() > bj2.getReshuffleCutoff()) {
                bj2.game(verbose:false);
                System.out.println(bj2.Deck.size());
            }
            bj2.reset();
            System.out.println(bj2.Deck.size());
            assert bj2.Deck.size() > 5;
            System.out.println(x:"*** Cutoff Works! ***\n");

        }

        System.out.println(x:"*** Done testing Blackjack! ***\n");
    }

    Run | Debug
    public static void main(String[] args) {

        blackjackTests();
    }
}
```

```
GAME BEGIN
Player Initial Hand: [10, 10] : 20
Dealer Initial Hand: [10, 6] : 16
Player Plays...
Player Ending Hand: [10, 10] : 20
Player did not bust! That's crazy!
Dealer Plays...
Dealer Ending Hand: [10, 6, 10] : 26
Dealer busted, Player Wins!
true == true
*** Player Turn Works! ***

false == false
*** Dealer Turn Works! ***

[] == []
[] == []
*** Reset Works! ***

2 == 2
2 == 2
*** Deal Works! ***

41
36
30
25
20
16
12
6
52
*** Cutoff Works! ***

*** Done testing Blackjack! ***
```

First the game itself was tested, to show initial hands and ending hands and the outcome of the game. To verify playerTurn() and dealerTurn(), I first created a boolean variable that would store the boolean returned by both methods after the outcome of their turns. Then, I wrote an if-else statement, which checked if the total value of cards in hand is greater than 21. If the hand had more than 21 cards, then it should return false, otherwise it should return true (since that is how the turn methods should run).

For reset(), I checked to see if both hands of the players were cleared, and printed the results and looked for an empty array of cards, hence the [].

For the deal method, I created an assert statement that sees if the hand of a player/dealer has two cards, since two cards should be dealt out at the beginning of the game. I printed their values just to confirm on the terminal.

Lastly, to check if the Deck would shuffle after it fell below the threshold, I ran a while loop for as many games until the amount of cards in the deck went below the cutoff counter, which was 10 for this test. Printing out the deck count not only helped show whether or not the Deck was being reset (after it fell below 10 cards), but also the fact that the Deck was not being reset after each individual game, and instead carrying over for the proceeding games. I ran reset one more time and printed the # of cards in the Deck, which increased back to52, showing that the deck was reshuffled.

```
GAME 1
======
GAME BEGIN
Player Initial Hand: [10, 7] : 17
Dealer Initial Hand: [6, 11] : 17
Player Plays...
Player Ending Hand: [10, 7] : 17
Player did not bust! That's crazy!
Dealer Plays...
Dealer Ending Hand: [6, 11] : 17
Dealer did not bust! That's crazy!
Both Players were equally close to 21, no one wins!

GAME 2
======
GAME BEGIN
Player Initial Hand: [2, 9] : 11
Dealer Initial Hand: [10, 11] : 21
Player Plays...
Player Ending Hand: [2, 9, 8] : 19
Player did not bust! That's crazy!
Dealer Plays...
Dealer Ending Hand: [10, 11] : 21
Dealer did not bust! That's crazy!
Dealer was closer to 21, Dealer wins!

GAME 3
======
Card count below the cutoff, SHUFFLING!
GAME BEGIN
Player Initial Hand: [4, 10] : 14
Dealer Initial Hand: [4, 8] : 12
Player Plays...
Player Ending Hand: [4, 10, 7] : 21
Player did not bust! That's crazy!
Dealer Plays...
Dealer Ending Hand: [4, 8, 10] : 22
Dealer busted, Player Wins!
```

Output of Running the game(true) Method 3 Times:

This is the output of three different Blackjack games from the Blackjack.java class. There are some additional prints implemented for the sake of narrating what happens in the game, such as saying whether or not someone didn't bust, and the card count went below the cutoff. These Strings would not print if (boolean verbose) was set to false.

Each game prints "GAME BEGIN" for organization, and runs reset() and deal() then prints both players' starting hands. To symbolize the Player's turn being played, it also prints "Player Plays…", then prints its ending hand. An if statement checks if the player's hand ended greater than 21, and since they didn't the game prints "Player did not bust! That's crazy" and moves on. Otherwise, then they would have busted and the game would end from there. The same logic follows for the Dealer's turn. As seen in Game 3, there is no "Dealer did not bust! That's crazy!" as they did in fact, bust. In games 1 and 2, the dealer didn't, so another if else statement processed whoever ended closer to 21,

and printed that outcome. In game 2, the dealer was on 21, and so the program printed the dealer as the winner. In game 1, neither player was closer nor farther to 21, therefore it states that fact.

Simulation:

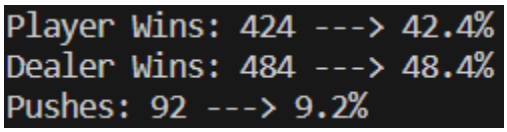
This is the output that is printed out in the terminal when Simulation.java is run. The output presents the number of wins for each player, as well as the percentages for the total number of dealer wins, player wins, and ties (pushes). As shown in the images, the dealer (the house) has a higher winning percentage than the player. To create this output, I created a for loop (shown in the right image) that iterated 1,000 times. Within the for loop, I had an if-else statement that checked whether the result of one game was the player, dealer, or a tie. If, for example, it was a player, then I would increment the playerWins by one. I would do this until the game was played 1000 times. To get the percentages, I divided the total number of wins by 1000.0 (.0 to convert to a double) and multiplied by 100 to get into a percentage.

```
public class Simulation {

    public static void simulation() {
        int pw = 0, dw = 0, p = 0;
        Blackjack bj1 = new Blackjack();
        for (int n = 0; n < 1000; n++) {
            int gameend = bj1.game(false);
            // System.out.println(gameend + " pw: " + pw + " dw: " + dw + " p: " + p );
            switch (gameend + 2) {
                case 1:
                    dw++;
                    break;
                case 2:
                    p++;
                    break;
                case 3:
                    pw++;
                    break;
            }
        }
        System.out.println("Player Wins: " + pw + " ---> " + pw / 1000.0 * 100 + "%");
        System.out.println("Dealer Wins: " + dw + " ---> " + dw / 1000.0 * 100 + "%");
        System.out.println("Pushes: " + p + " ---> " + p / 1000.0 * 100 + "%");
    }

    Run | Debug
    public static void main(String[] args) {

        simulation();
    }
}
```



```
Player Wins: 424 ---> 42.4%
Dealer Wins: 484 ---> 48.4%
Pushes: 92 ---> 9.2%
```

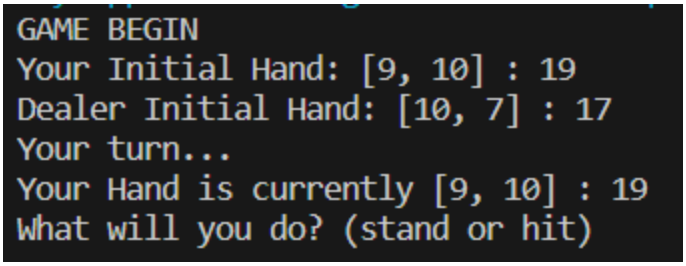
Discussion

Overall, looking at the results of the Blackjack game, they do make sense. As shown from the images that I got in the Simulation.java, the house always wins, which can be held true mostly because the player always plays first before the dealer plays. If the player busts in the first round of games, the dealer automatically wins without needing to play. Even if the player does not bust, the dealer has the opportunity to think and make a decision based on the player's play, making its chances of winning higher than the player's.

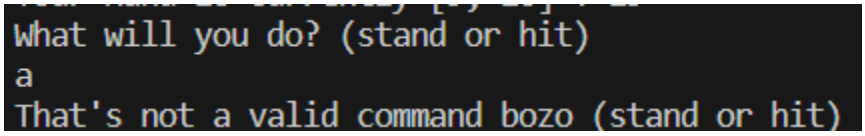
Extensions

playerTurnInteractive()

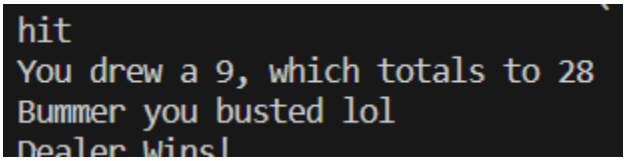
Adding the playerTurnInteractive() class allows the player to use the terminal to input whether or not they want to hit or stand during their turn. This implements the java.util.Scanner subclass. A Scanner object *input* is created at the constructor of the Blackjack class, which will be used for asking and storing whatever the player inputs in the code.



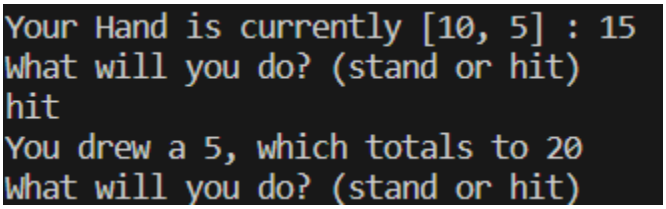
The game begins as it usually does normally, instead of referring to the player as you, the person who's operating the terminal. A while loop begins to repeat until the player stands, or has enough cards to equal or be greater than 21.



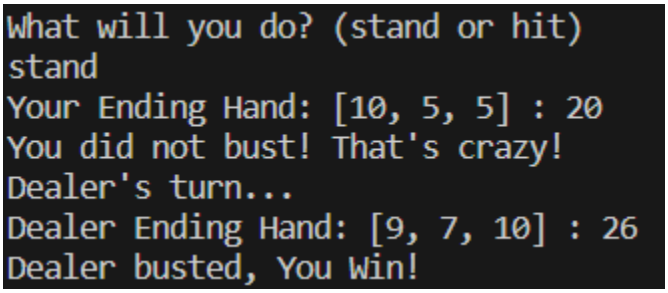
To account for mistakes in spelling or accidentally typing in an invalid command, a while loop plays until either "stand" or "hit" is sent through .nextLine(). (This is made non case sensitive through .equalsIgnoreCase("stand") and .equalsIgnoreCase("hit")) In this case, typing "a" is an invalid command, so the console restates the available commands, and gives another chance to retype a proper command.



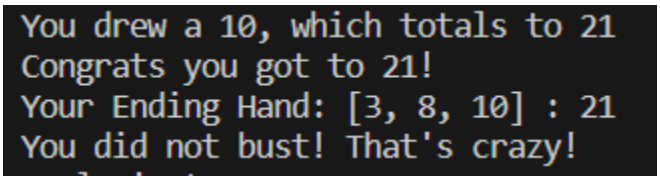
Typing "hit" adds the next card from the deck to the player's hand, and the game prints out what the card was and what it totaled to. Since they went over 21, they busted, and so the while loop ends and the turn, and game finishes in the player's loss.



If they do not bust, the console will keep asking for their decision until they bust, hit 21, or stand.



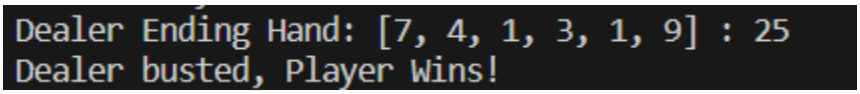
Standing ends the player's turn, and the console restates their ending hand, and that they didn't bust. The dealer's turn plays out as it normally would, and the rest is as exactly as it is in the original game().



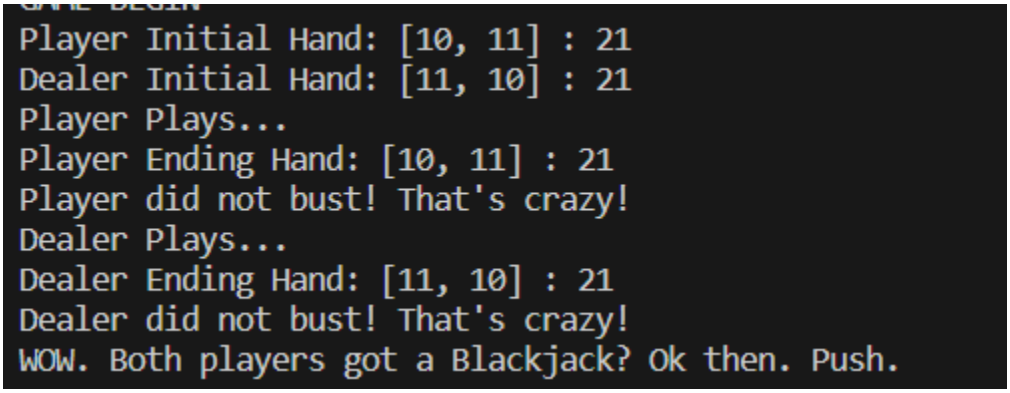
Hitting to a 21 also instantly ends the player's turn, while also leaving a message.

Ace + Real Blackjack

In traditional Blackjack, the Ace can have a value of either 1 or 11, depending on which is more advantageous. In the code, I programmed it so that every time a player would draw an 11 card, if that would make them bust, the card would instead have a value of 1.



Another rule is a real Blackjack, in which a hand of 2 cards totaling 21 beats a 21 with more than 2 cards when evaluating the winning hand. In this case, many outcomes may occur:



Both players getting a Blackjack is a push.

```
GAME BEGIN
Player Initial Hand: [3, 10] : 13
Dealer Initial Hand: [10, 3] : 13
Player Plays...
Player Ending Hand: [3, 10, 8] : 21
Player did not bust! That's crazy!
Dealer Plays...
Dealer Ending Hand: [10, 3, 8] : 21
Dealer did not bust! That's crazy!
Wow both players got to 21, that's crazy! That's a push tho.
```

Both players getting 21 is a push.

```
if (Player.size() == 2) {
    if (verbose == true) {
        System.out.println(
                "Even though both players got 21, the Player hit a Blackjack, so they win!");
    }
    return 1;
} else if (Dealer.size() == 2) {
    if (verbose == true) {
        System.out.println(
                "Even though both players got 21, the Dealer hit a Blackjack, so they win!");
    }
    return -1;
```

If the player got a Blackjack, even though the Dealer drew 21 with more than 2 cards it would still be the player's win, and vice versa.

```
Player Wins: 454 ---> 45.4%
Dealer Wins: 465 ---> 46.5%
Pushes: 81 ---> 8.1%
```

The percentages from the Simulation class seem a lot closer together, but the dealer still has higher chances of winning, just not as significant.

6 Decks

A for loop adds the standard 52 cards to the Card Deck 6 times, with the reshuffle cutoff now being 52 (the size of 1 card deck)

```
Player Wins: 394 ---> 39.4%
Dealer Wins: 521 ---> 52.1%
Pushes: 85 ---> 8.5%
```

This significantly lowered the chances of the player winning.

References/Acknowledgements

I asked for help from my sister Claire, who previously took this class. I looked at her report because I have no clue how to format reports and what to include as information, so I needed an example. I also asked for input from Prof. Lage and Prof. Bender about what to include on this report.