MaddyPlacik_CS231_Project1

Project Description

For this project, we were instructed to simulate a game of Blackjack. In order to do this, we created multiple classes to represent the player/dealer hands, the cards, the deck of cards, the game Blackjack itself, and a simulation of the game. The Hand, Card, and Deck classes were used in the Blackjack class directly, and the simulation class just used the Blackjack class. The simulation class was used to simplify the game (representing game results as 0, 1, and -1) – allowing for multiple efficient iterations.

Solution Explanation

In this project, the Card class held an integer value ranging from 1-10. Card objects were then stored in both Hand and Deck objects, which were both Array List objects.

There were several complicated components to this project - one of which included the shuffle function. The shuffle function allowed us to shuffle our deck of cards in a random manner. The shuffle function was contained in the Deck class and worked by iterating through each card of an un-shuffled deck object - placing the card into a random index of a new deck object. As the card is placed into the new deck object, it is also removed from the un-shuffled deck. Below is my shuffle function laid out:

```
public void shuffle(){
    ArrayList<Card> newDeck = new ArrayList<Card>();
    for (int i = 0 ; i < 52 ; i++){
        int index = randy.nextInt( 52 - i);
        newDeck.add(this.deck.remove(index));
        //this.deck = newDeck;
    }
    this.deck = newDeck ;
}
```

In order to create a functioning Blackjack class, I created instance variables of a deck, player hand and a dealer hand. In order to create an efficient simulation of my game, I created a function called numClass within my Blackjack class. This function returned a value of 1 for player wins, a value of -1 for dealer wins and a value of 0 for ties. In order to create this function correctly, I needed to input every possible game scenario. In order for the player to win, they needed to have a total card value of 21 or less and needed to have a total card value greater than the dealer's. The same holds for the dealer. A tie would occur if both hands had the same total value, equal to or under 21. I represented each scenario with the following lines of code:

```
if ( playerTurn1 == false ){
    return -1;
}

else if ( dealerTurn1 == false && playerTurn1 == true){
    return 1;
}

else if ( this.playerHand.getTotalValue() > this.dealerHand.getTotalValue() ){
    return 1;
}

else if ( this.playerHand.getTotalValue() < this.dealerHand.getTotalValue() ){
    return -1;
}

else {
    return 0;
}
```

I then used the numClass function in my Simulation class to simulate 1000 games of Blackjack. I calculated the number of player wins, dealer wins and ties by saving these totals as integer value variables. I set the initial value of the wins to zero and then ran through 1000 games. As each game runs, the main function checks the return value of the numClass function. Depending on the return value, 1 would be added to either the tie,

player win or dealer win variables - essentially creating a tally system. Please see the code below.

```
for (int i = 0 ; i < 100 ; i++ ){
    Blackjack games = new Blackjack();
    int val = games.numClass();

    if (val == 0){
        push += 1.0;
    }
    else if (val == 1){
        playerWins += 1.0;
    }
    else if (val == -1){
        dealerWins += 1.0;
    }

}
```

I believe that 1000 games is sufficient to provide an accurate estimate of the win percentage for the player and dealer. These values change with every 1000 games but the player typically wins about 40-60% of the games and the dealer typically wins 30-50% of the games. The percentage of games that result in ties usually lies within 0-15%.

Running the game about 90,000 times (using 6 decks of cards) will allow the dealer to always win more games than the player.

Extensions

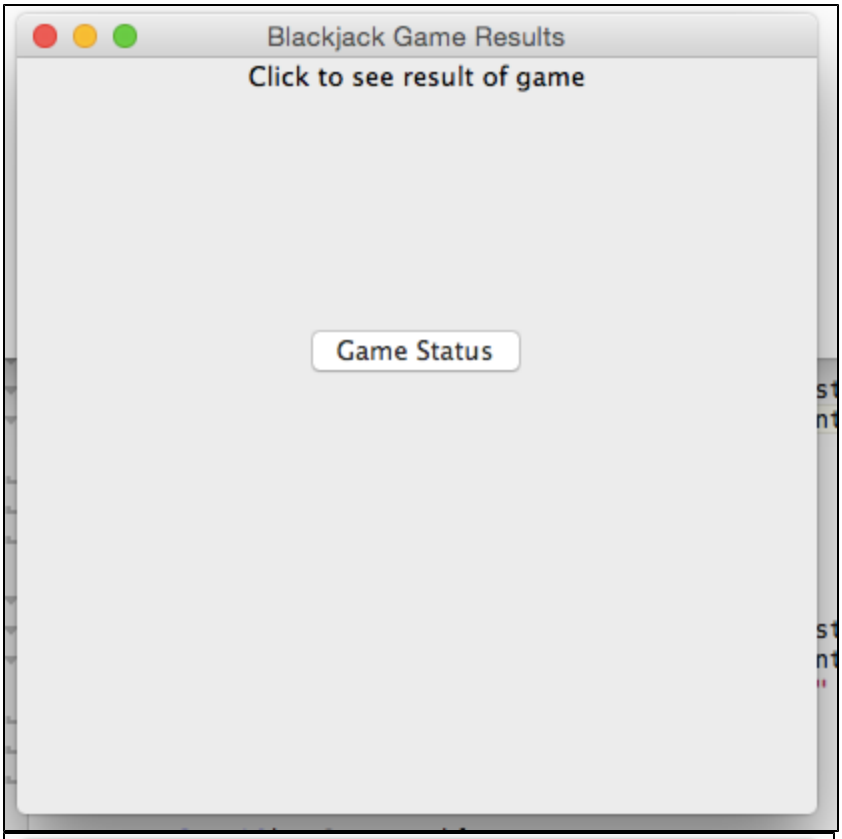
I chose to complete two extensions for this project. My first extension used 6 decks of cards instead of 1 and shuffled the complete deck after 1 deck remained. This extension is available in the files with "_ext" in their names.

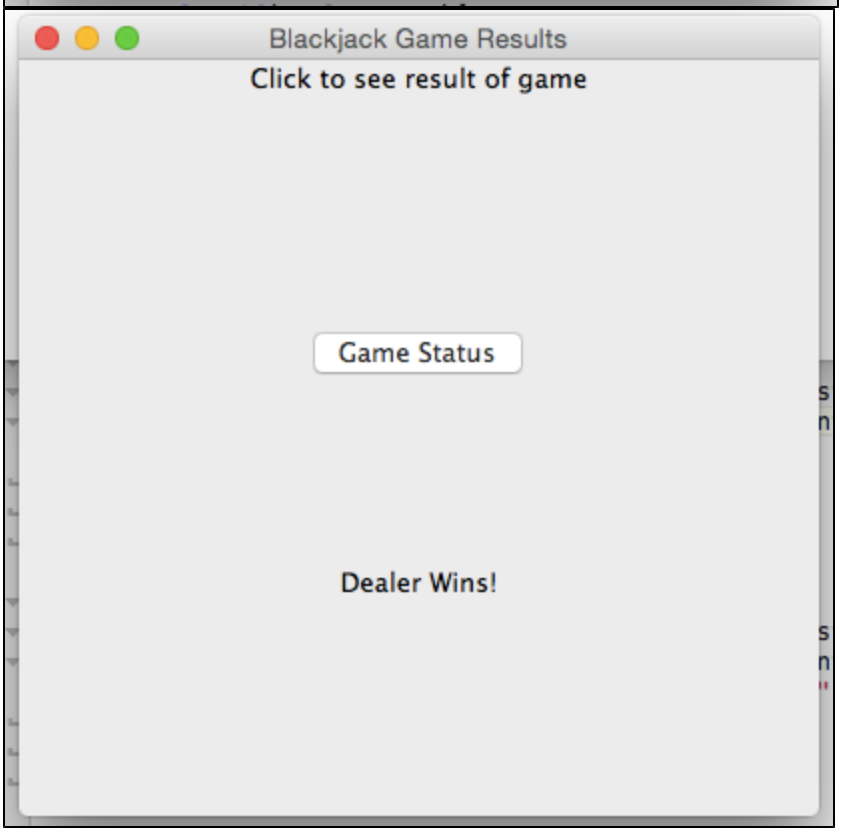
I built these 6 decks of cards as one deck using the following code:

```
//builds 6 decks
public void build6(){
    for (int j=1; j<10; j++){
        for (int i=0; i<24; i++) {
            Card card1 = new Card(j);
            this.deck.add(card1);
        }
    }

    for (int i=0; i<96; i++){
        Card card2 = new Card(10);
        this.deck.add(card2);
    }
}
```

My second extension created a visual representation of the game. In order to do this, I created another file titled Window.java. This file created a window with prompt "Click to see result of game" and a button with the text "Game Status." When this button is clicked, the winner of the game is revealed. Every time the main method of this class runs, a single Blackjack game is executed. Below is a picture of the window created for a single game before and after the button is clicked:





This method called the numClass function of the Blackjack class in order to obtain game results.

Conclusion

Now that I have finished this project, I have a much better understanding of classes, methods, and variables. I've come to a better understanding of how to create and use objects of various classes throughout different classes and retrieving functions via these objects. I have also learned a

lot about array lists and their unique capabilities to store information. It was interesting to see each component of the project come together to form a cohesive game. I don't think I understood the importance of each function within each class until I saw the impact that they had on the overall system. Creating a game of Blackjack seemed quite overwhelming at first, but by separating the game into different units (classes) the project seemed much more manageable and comprehensive.