Project 1

PROJECT 1

In this project, our overall task was to create multiple classes to represent different parts of whole towards a game of Blackjack. Implementing classes for the cards, the hand, the deck, and the game Blackjack helped us begin to understand the format and dictation of Java; a language completely new to us. Using Java, we are asked to simulate the game many times and study the game of Blackjack and its patterns. Overall, this lab was a introduction to Java classes through the creation of a Blackjack game simulator.

Task 1. In this task, we were asked to create a Card class that basically generates a card with a value between 1-10.

Task 2. In this task, we created a Hand class to hold a set of cards. To do this, we had to import and ArrayList and add the values of the cards to an ArrayList Hand. Basically, we created a list of a specific number list cards and called it a hand. Within this class, we also created some useful methods including getTotalValue() and public int size(). The getTotalValue() method returns the total value of the cards within the hand and the size() method returns the number of cards in the hand.

Task 3. Next, were created a class called Deck that modeled a standard 52-card deck. Within the deck, we had to create a void build() method that added 52 cards to the ArrayList of Deck making sure to include 4 sets of cards with the values 1-9 and 16 cards with the value 10. To do this, I used nested for loops. The code for this looked like this:

```
    public void build(){
        this.deck.clear();
        for( int i = 0; i < 4; i++){
            for( int j = 1; j <= 9; j++){
                this.deck.add(new Card(j));
            }
            for(int k = 0; k < 4; k++){
                this.deck.add(new Card(10));
            }
        }
    }
```

Also, in the Deck classes we were also asked to make a void shuffle() method. To tackle this, I decided to create an entirely new deck, clear the old original deck and randomly add the new deck's cards into the old deck's list. With some assistance from a good friend and talented CS student CP Majgaard, I was able to use something called System.currentTimeMillis(). This is a random generator that generates its values from a clock that has been running from a specific start date. It creates arbitrary random values and using this I was able to randomly add the new deck into the old deck to make a shuffle method. Here is the code snippet:

```
public void shuffle(){
    // assistance from CP and stack-overflow
    Random generator = new Random(System.currentTimeMillis());
    Deck deck2 = new Deck();
    this.deck.clear();
    for( int i = 0; i < 52; i++){
        this.deck.add(deck2.pick(generator.nextInt(deck2.size())));
    }


}
```

Task 4. This task was the most important. In this task, we initialized a Blackjack class that had a Deck, a Hand for the dealer and for the player, and some type of scoring system. Two important methods I created were called boolean playerTurn() and boolean dealerTurn(). Both of these method set rules for when the player or dealer would ask for another card when he/she would stay with the hand they already had. Also, I had to take into account that when a person had a hand with a value greater than 21, the game is over. Here is the playerTurn() code:

```
  public boolean playerTurn(){
      while(this.playerHand.getTotalValue() <16){
          //System.out.println("Player wants to hit");
          this.playerHand.add(deck.deal());
      }
      if( this.playerHand.getTotalValue() > 21 ){
          return false;
      }
      else{
          return true;
      }
  }
```

The main method of this Blackjack class was very intricate. It dealt both hands and then called playerTurn(). From there, the player could hit, stay, or bust. If the player hit/stayed, the game would eventually carry on to the dealers move and dealerTurn(). Same as before, the dealer could either hit, stay or bust. If either of the two players busted, the game was over. I used a lot of for loops and conditionals in my main method to compare

hand values and then multiple print statements to show the winner and the value of his/her hand. Also, using my toString() method allowed me to see the status of the game throughout each simulation.

Task 5. This task just wanted text files of 3 different game simulations. Here are my three games:

mygame1.txt

```
Player is dealt: 8
Dealer is dealt: 11
Player Hand contents: 2, 6: 8
Dealer Hand contents: 3, 8: 11
Player wants to hit
Player wants to hit
Player wants to hit
Player will stay with: 19
Dealer wants to hit
Dealer will stay with: 21
Dealer wins with hand of: 21
Player Hand contents: 2, 6, 1, 1, 9: 19
Dealer Hand contents: 3, 8, 10: 21
```

mygame9.txt

```
Player is dealt: 16
Dealer is dealt: 11
Player Hand contents: 7, 9: 16
Dealer Hand contents: 4, 7: 11
Player will stay with: 16
Dealer wants to hit
Dealer wants to hit
Dealer wants to hit
Game is over! Dealer has busted and player wins!
Player Hand contents: 7, 9: 16
Dealer Hand contents: 4, 7, 4, 1, 10: 26
```

mygame10.txt

```
Player is dealt: 17
Dealer is dealt: 7
Player Hand contents: 10, 7: 17
Dealer Hand contents: 6, 1: 7
Player will stay with: 17
Dealer wants to hit
Dealer will stay with: 17
It is a PUSH!
Player Hand contents: 10, 7: 17
Dealer Hand contents: 6, 1, 10: 17
```

*Task 6. *This task asked to add another function to Blackjack that acted like a scoring system for that individual simulation. If the player won return a 1, if the dealer won return a -1, and if it was a push return a 0. I created a function called getScore() and here is what it looked like

```
public int getScore(){
    this.scoreBoard = 0;
    if ( this.dealerHand.getTotalValue() == this.playerHand.getTotalValue()){
        this.scoreBoard += 0;
    }
    else if(( this.dealerHand.getTotalValue() > this.playerHand.getTotalValue())
        && (this.dealerHand.getTotalValue()<=21)){
        this.scoreBoard -= 1;
    }
    else if(( this.playerHand.getTotalValue() > this.dealerHand.getTotalValue())
        && (this.playerHand.getTotalValue()<=21)){
        this.scoreBoard += 1;
    }

    else if( this.playerHand.getTotalValue() > 21){
        this.scoreBoard -= 1;
    }

    else if( this.dealerHand.getTotalValue() > 21){
        this.scoreBoard += 1;
    }

    return this.scoreBoard;
}
```

*Task 7. *This task asked to create a new class called Simulation that executes 1000 games of Blackjack and keeps track of the scores. Since my getScore() function was written for a single game and score only, I had to write a new function called Game() that both ran the game of Blackjack and kept track of all the wins,loses, and pushes. Then, in the main function of my Simulation class, I was able to make three integer object called pushes, playerWins, dealerWins, and then us conditionals and if statements to total all of the wins and pushes and print them. My simulation code without print statements looks like this:

```java
public class Simulation{

    public static void main( String[] args ){
        Blackjack b1 = new Blackjack();
        int score;
        int pushes = 0;
        int dealerWins = 0;
        int playerWins = 0;
        for( int i = 0; i<1000; i++ ){
            score = b1.Game();
            if(score == 0){
                pushes += 1;
            }
            else if(score == 1){
                playerWins += 1;
            }
            else if(score == -1){
                dealerWins += 1;
            }
        }
```

Then, I added print statements to give me values and then calculate the averages!
Here are the results of my overall simulation:

```
tonys-new-mbp:Project1 Tony$ javac Simulation.java
tonys-new-mbp:Project1 Tony$ java Simulation
Total number of Pushes = 165
Total number of Player Wins = 339
Total number of Dealer Wins = 496
Percentage of Pushes = 0.165
Percentage of Player Wins = 0.339
Percentage of Dealer Wins = 0.496
```

Discussion. *1000 games of Blackjack will give you a good estimate of the breakdown of ratios of wins and loses. It will depict around 45% dealer, 40% player, and 15% push. For me, itThese percentages are what I gathered from a few simulations, *BUT a larger the sample size is much better and more telling. And this is what I did in my extension!

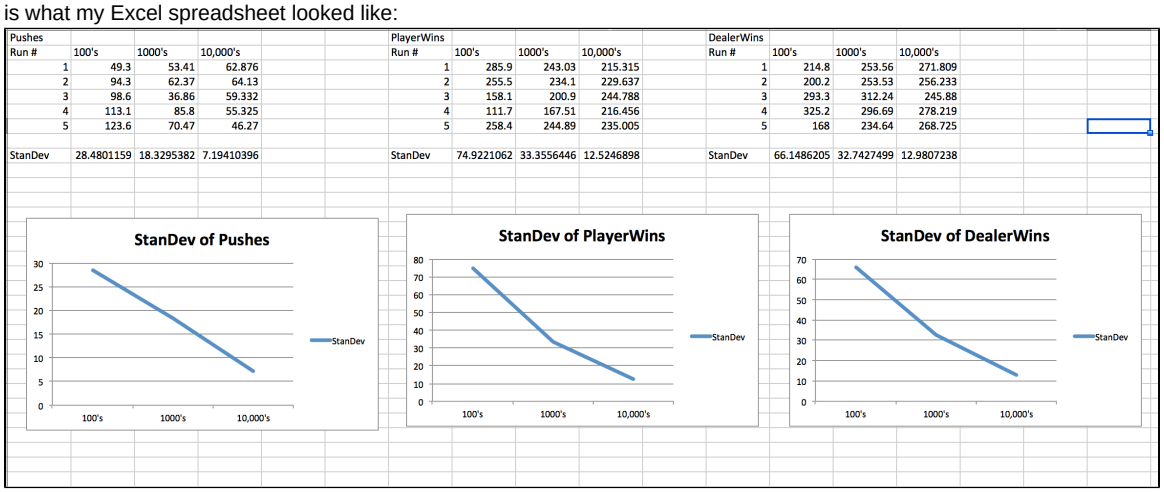
*Extension! *For this extension, I followed the suggested extension # 5. This extension asked me to look at different sample sizes and compare the standard deviations. Going into the extension, my hypothesis was that the larger the sample size, the smaller the deviation, and the more solid the results/percentages. So, to complete this extension, I created three new classes called Extension1a, 1b, and 1c. These three classes were striking similar to my Simulation class, except I used nested for loops to create averages. Here is a look at my Extension1c code:

```java
public class Extension1c{

    public static void main( String[] args ){
        Blackjack b1 = new Blackjack();
        int score;
        int pushes = 0;
        int totalPushes = 0;
        int dealerWins = 0;
        int totalDealer = 0;
        int playerWins = 0;
        int totalPlayer = 0;
        for( int i = 0; i<10; i++ ){
            for( int j =0; j <10000; j++){
                score = b1.Game();
                if(score == 0){
                    pushes += 1;
                }
                else if(score == 1){
                    playerWins += 1;
                }
                else if(score == -1){
                    dealerWins += 1;
                }
            }
            totalPushes += pushes;
            totalDealer += dealerWins;
            totalPlayer += playerWins;
```

At the end of the Extension1c main method, I totaled all the averages and then averaged the averages by using print statements and division. The next significant step in this rather large extension was to pool the data and compare standard deviations. I ran each extension class 5 times and

recorded all the data for pushes, playerWins, and dealerWins in an Excel spreadsheet. I then created a graph for each of the three classes. Here is what my Excel spreadsheet looked like:



Pushes			
Run #	100's	1000's	10,000's
1	49.3	53.41	62.876
2	94.3	62.37	64.13
3	98.6	36.86	59.332
4	113.1	85.8	55.325
5	123.6	70.47	46.27
StanDev	28.4801159	18.3295382	7.19410396

PlayerWins			
Run #	100's	1000's	10,000's
1	285.9	243.03	215.315
2	255.5	234.1	229.637
3	158.1	200.9	244.788
4	111.7	167.51	216.456
5	258.4	244.89	235.005
StanDev	74.9221062	33.3556446	12.5246898

DealerWins			
Run #	100's	1000's	10,000's
1	214.8	253.56	271.809
2	200.2	253.53	256.233
3	293.3	312.24	245.88
4	325.2	296.69	278.219
5	168	234.64	268.725
StanDev	66.1486205	32.7427499	12.9807238

As predicted, with the larger sample size, the standard deviation decreased and the data became more reliable. I only did one extension because this extension had so many parts.

Conclusion: In conclusion, I learned a lot about Java and how it is organized and written. I successfully created Blackjack game simulator and all the classes that go along with it. Through simulations I was able to find ratios and probabilities of the game. Overall, this project made me a lot more comfortable with Java and how it use object oriented code to organize itself.

Who Helped Me:

I received help from TA Kellam, Prof Bruce Maxwell, fellow CS 231 student Julia Saul, and CP Majgaard.