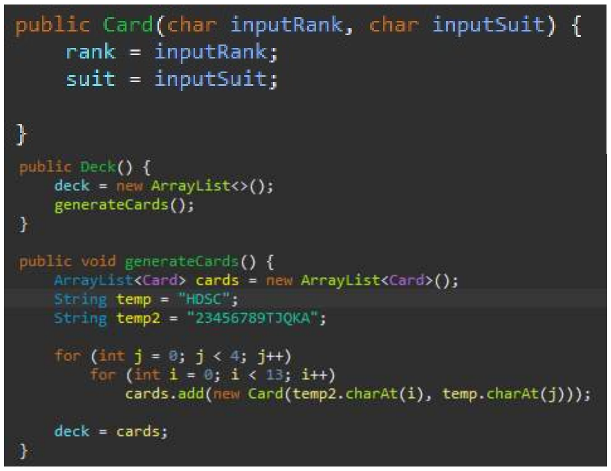
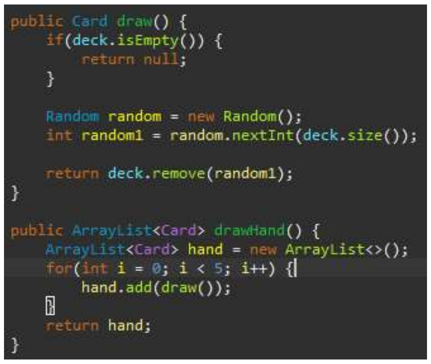
Piece 5
Java Poker Project

This java project was created by making the Card object first. My logic was that in order to have a Card, the card would be a suit and a rank. Therefore, to make a Card, the parameters that it requires is Card(suit, rank). After the Card was object was created, I had to create a deck that contained 52 cards. I knew that the deck had to be 52 cards and that each card would have a rank and a suit. I created a method named generateCards() that would take the Card and give it two characters ("HDSC" and "23456789TTQKA"). This would allow me to have the deck filled with Card objects that contained the rank and the suit.



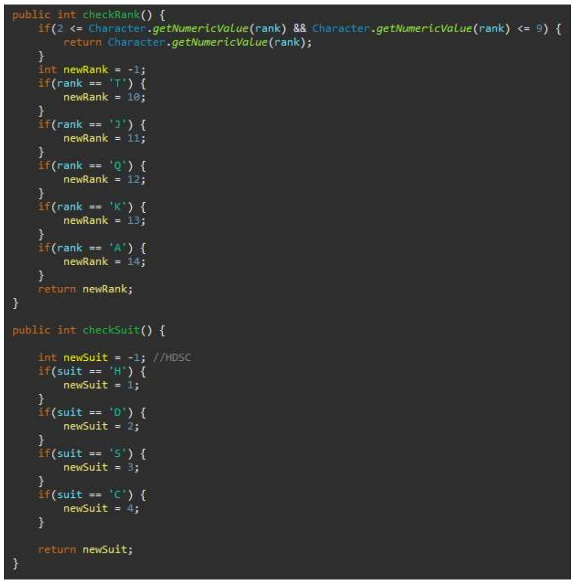
```java
public Card(char inputRank, char inputSuit) {
    rank = inputRank;
    suit = inputSuit;

}
public Deck() {
    deck = new ArrayList<>();
    generateCards();
}

public void generateCards() {
    ArrayList<Card> cards = new ArrayList<Card>();
    String temp = "HDSC";
    String temp2 = "23456789TJQKA";

    for (int j = 0; j < 4; j++)
        for (int i = 0; i < 13; i++)
            cards.add(new Card(temp2.charAt(i), temp.charAt(j)));

    deck = cards;
}
```

Next, I went ahead and created a method to draw(). This method would take a card from the deck, remove it from the deck and put it into my hand. Once this was working, I was able to create the drawHand() method that would call the draw method five times and put the cards into an ArrayList named hand. An idea that I had that would make my life a lot easier was to sort the ArrayList hand prior to checking the hand. This is because if I have a straight for example, then I know that the second card in my hand would be the first card plus 1. Another example is that if I have a royal flush and the hand is sorted, I know that the 3rd card in my hand (otherwise known as the middle card) would have to be a Queen.



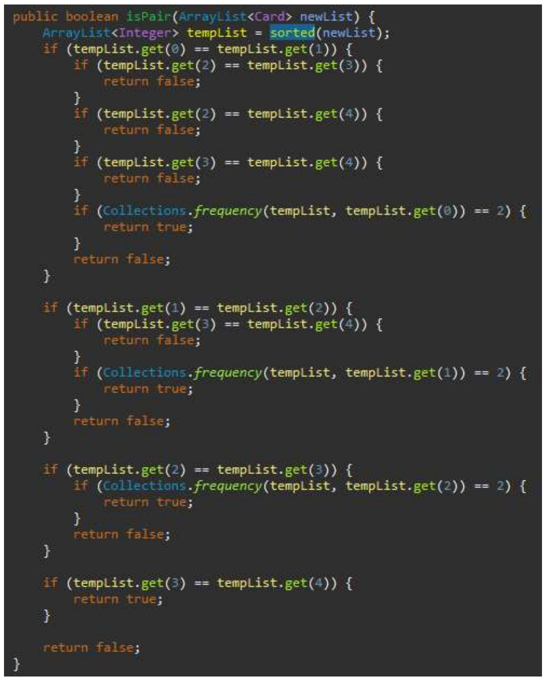
```java
public Card draw() {
    if(deck.isEmpty()) {
        return null;
    }

    Random random = new Random();
    int random1 = random.nextInt(deck.size());

    return deck.remove(random1);
}

public ArrayList<Card> drawHand() {
    ArrayList<Card> hand = new ArrayList<>();
    for(int i = 0; i < 5; i++) {
        hand.add(draw());
    }
    return hand;
}
```

However, the issue was that if I had a card and its rank was 2 and another card with the rank of Q, I couldn't compare an integer to a character that way. Therefore, inside of the Card class, I had the Card rank and suit change from 'A' to 14, 'K' to 13, and so forth. Since the Card ranks and the suits were now able to be compared to each other, I was able to start creating methods like isPair(). IsPair will sort the hand prior to checking anything. Then, I check if the first two cards are the same and the third and fourth cards. If they are the same, then it is a twoPair instead of one pair. If the first two cards are not the same, it would check the 3rd and 4th card, and then the fourth and the fifth card.



```java
public int checkRank() {
    if(2 <= Character.getNumericValue(rank) && Character.getNumericValue(rank) <= 9) {
        return Character.getNumericValue(rank);
    }
    int newRank = -1;
    if(rank == 'T') {
        newRank = 10;
    }
    if(rank == 'J') {
        newRank = 11;
    }
    if(rank == 'Q') {
        newRank = 12;
    }
    if(rank == 'K') {
        newRank = 13;
    }
    if(rank == 'A') {
        newRank = 14;
    }
    return newRank;
}

public int checkSuit() {

    int newSuit = -1; //HDSC
    if(suit == 'H') {
        newSuit = 1;
    }
    if(suit == 'D') {
        newSuit = 2;
    }
    if(suit == 'S') {
        newSuit = 3;
    }
    if(suit == 'C') {
        newSuit = 4;
    }

    return newSuit;
}
```

Then, I would check twoPair() because the logic of it was similar to isPair(). I would check if the first two cards are the same and if the third and fourth cards are the same. If the first two cards are not the same, I would check the second card with the third card and then the fourth card with the fifth card. However, using Collections.frequency() saved my life because I was able to check how many times that card is in the hand. Therefore, I knew that if the first card and the second card were a pair but, I was receiving a frequency of 3 then I knew that I had three of a kind rather than twoPair.



```java
public boolean isPair(ArrayList<Card> newList) {
    ArrayList<Integer> tempList = sorted(newList);
    if (tempList.get(0) == tempList.get(1)) {
        if (tempList.get(2) == tempList.get(3)) {
            return false;
        }
        if (tempList.get(2) == tempList.get(4)) {
            return false;
        }
        if (tempList.get(3) == tempList.get(4)) {
            return false;
        }
        if (Collections.frequency(tempList, tempList.get(0)) == 2) {
            return true;
        }
        return false;
    }

    if (tempList.get(1) == tempList.get(2)) {
        if (tempList.get(3) == tempList.get(4)) {
            return false;
        }
        if (Collections.frequency(tempList, tempList.get(1)) == 2) {
            return true;
        }
        return false;
    }

    if (tempList.get(2) == tempList.get(3)) {
        if (Collections.frequency(tempList, tempList.get(2)) == 2) {
            return true;
        }
        return false;
    }

    if (tempList.get(3) == tempList.get(4)) {
        return true;
    }

    return false;
}
```

The isThree() method would check to see if the first three cards are equal to each other. If they are, then it would check if the fourth and fifth card are a pair or not. If they are, then it would be three of a kind. However, if it is not the first three cards, we could check the middle three cards out of the five in the hand. If they are equal and had a frequency of three, then we have a three of a king. Lastly, I would check the last three cards to see if they are equal to each other which in that case we would have a straight as well.

The isFour() method would work in a similar fashion. If the first card is equal to the fourth card, then it is a four of a kind as long as the fifth card is not equal as well. I did this by checking the frequency of the first card equal to 4. If it was anything else, it was not a four of a kind. I would then check the fifth card for a frequency of four that would determine whether or not it was a straight.

In order to check a straight, I created the straight() method that would check if the hand is in order. Therefore, I would check to see if the fifth card is a 14 (Ace). If it is 14, then I would check to see if the first card is a two. Then I would check if the second card is the first card plus one. If that holds true for the rest of the hand, then it is a straight. Otherwise, if the fifth card is not an ace, then I would check to see if the second card is the first card plus one all the way through the hand. If it holds true, then it is a straight.

A flush was very easy to create. I would call the sortSuit() method so that I could have all of the suits change to numerical values (1,2,3,4,5). If the suit in the hand has a Collections.frequency of 5, the it would be a flush. There is nothing else to check in this method other than the frequency of a single card in the hand. To check whether the hand is a full house, we would take bits from both isPair() and isThree(). We would check to see if the first three cards are equal and that the last two cards are equal however, we made sure that the third card is not equal to the fourth card. If that did not work, then we would check to see if the third, fourth, and fifth cards are equal to each other. If they are, then it is not a full house. Otherwise, if it is all true, it would be a full house.

To check if the hand is a straight flush, I would use both methods that I created prior to check. I would check the hand to see if it is a straight and then a flush. If both of those methods return true, then it is a straight flush. To check if the hand is a royal flush, you would check to see if the hand is not a straight flush. If it is a straight flush, you would check to see that the third card is a Queen. If it is, then it is a straight flush.

Overall, this is one of the hardest projects that I have done at Stockton. I believe that this took me a few weeks to finish however, I am very proud of the results that I received. I didn't know as much about Object oriented programming as I though and I was able to teach myself a few things that I didn't know about before such as Collections.frequency. I found it very difficult to compare the actual Card objects to each other. It took me a while until I figured out that I could just change the characters to numerical values and then compare them that way. Once I was able to figure that out, the rest of the project was relatively easier that the initial set up. This is a project that I plan on working on a little more after this course so that I can include it on my resume. I don’t think that I’ve ever worked so hard on a project other than this one.