Noah Streveler

Assignment 2

End Output

```
>Player3's Turn

>Player3 got the 9 of Clubs
9 of Spades and 9 of Clubs have been removed!
Player3 had the Old Maid, they lose!
```

```
import java.util.ArrayList;
import java.util.Collections;

/**
 * An object of type Hand represents a hand of cards.  The
 * cards belong to the class Card.  A hand is empty when it
 * is created, and any number of cards can be added to it.
 */
public class Hand {
    private ArrayList<Card> hand;   // The cards in the hand.

    public Hand() {
        hand = new ArrayList<Card>();
    }

    public void clear() {
        hand.clear();
    }

    /**
     * Add a card to the hand.  It is added at the end of the current hand.
     * @param c the non-null card to be added.
     * @throws NullPointerException if the parameter c is null.
     */
    public void addCard(Card c) {
        if (c == null)
            throw new NullPointerException("Can't add a null card to a hand.");
        hand.add(c);
    }

    /**
     * Remove a card from the hand, if present.
     * @param c the card to be removed.  If c is null or if the card is not in
     * the hand, then nothing is done.
     */
    public void removeCard(Card c) {
        hand.remove(c);
    }
```

```
    /**
     * Remove the card in a specified position from the hand.
     * @param position the position of the card that is to be removed, where
     * positions are starting from zero.
     * @throws IllegalArgumentException if the position does not exist in
     * the hand, that is if the position is less than 0 or greater than
     * or equal to the number of cards in the hand.
     */
    public void removeCard(int position) {
        if (position < 0 || position >= hand.size())
            throw new IllegalArgumentException("Position does not exist in hand: "
                    + position);
        hand.remove(position);
    }

    /**
     * Returns the number of cards in the hand.
     */
    public int getCardCount() {
        return hand.size();
    }

    /**
     * Gets the card in a specified position in the hand.  (Note that this card
     * is not removed from the hand!)
     * @param position the position of the card that is to be returned
     * @throws IllegalArgumentException if position does not exist in the hand
     */
    public Card getCard(int position) {
        if (position < 0 || position >= hand.size())
            throw new IllegalArgumentException("Position does not exist in hand: "
                    + position);
        return hand.get(position);
    }

    /**
     * Sorts the cards in the hand so that cards of the same suit are
     * grouped together, and within a suit the cards are sorted by value.
     * Note that aces are considered to have the lowest value, 1. --- sorting is
similar to "selection sort"
     */
    public void sortBySuit() {
        ArrayList<Card> newHand = new ArrayList<Card>();
        while (hand.size() > 0) {
            int pos = 0;  // Position of minimal card.
            Card c = hand.get(0);  // Minimal card.
            for (int i = 1; i < hand.size(); i++) {
                Card c1 = hand.get(i);
                if ( c1.getSuit() < c.getSuit() ||
                        (c1.getSuit() == c.getSuit() && c1.getValue() < c.getValue())
) {
                    pos = i;
                    c = c1;
                }
            }
```

```
            hand.remove(pos);
            newHand.add(c);
        }
        hand = newHand;
    }

    /**
     * Sorts the cards in the hand so that cards of the same value are
     * grouped together.  Cards with the same value are sorted by suit.
     * Note that aces are considered to have the lowest value, 1.
     */
    public void sortByValue() {
        ArrayList<Card> newHand = new ArrayList<Card>();
        while (hand.size() > 0) {
            int pos = 0;  // Position of minimal card.
            Card c = hand.get(0);  // Minimal card.
            for (int i = 1; i < hand.size(); i++) {
                Card c1 = hand.get(i);
                if ( c1.getValue() < c.getValue() ||
                        (c1.getValue() == c.getValue() && c1.getSuit() < c.getSuit())
) {
                    pos = i;
                    c = c1;
                }
            }
            hand.remove(pos);
            newHand.add(c);
        }
        hand = newHand;
    }

    public boolean equals(int num1, int num2) {
       if(num1 == num2)
             return true;

       return false;
    }

    public Hand removePairs(Hand group) {
             for(int i = 0; i < group.getCardCount() - 1; i++) {
                   boolean pair = equals(group.getCard(i).getValue(),
group.getCard(i+1).getValue());
                   if(pair) {
                         System.out.println(group.getCard(i) + " and " +
group.getCard(i+1) + " have been removed!");
                         group.removeCard(i);
                         group.removeCard(i);
                         i--;
                   }
             }
       return group;
    }

}//end of class Hand
```

```
import java.util.*;
import java.time.*;

public class OldMaid {
   public static void main(String[] args) {
            GameController controller= new GameController();
            controller.runGame();
   }
}

class IOHandler implements IGameView{
        Scanner  sc = new Scanner(System.in);
        char input;
        int players;
        int grabbed;

        @Override
        public void display(String message) {
                System.out.println(message);
        }

        @Override
        public int grabCard(int lowerbound, int upperbound) {
                while(true) {
                        System.out.println("\n>Which Card would you like to grab(" + lowerbound + "-"
+ upperbound + ")?");
```

```
                        grabbed = sc.nextInt();
                        if(grabbed < lowerbound || grabbed > upperbound) {
                                System.out.println("The Card you tried to grab was outside the range,
please try again");
                        }
                        else {
                                return grabbed - 1;
                        }
                }

        }

        @Override
        public int getPlayers(String prompt) {
                System.out.println(prompt);
                players = sc.nextInt();
                if(players < 2 && players > 12) {
                        System.out.println("I'm sorry, but the option that you chose is not available,
please try again");
                        System.exit(0);
                }
                //input = JOptionPane.showInputDialog(prompt);
                return players;
        }

}

class GameController implements IGameControl{  //game model + game control
        IGameView h = new IOHandler();
```

```
public Deck deck;



public GameController(){
        init();
}

@Override
public void init(){
        String str = "Welcome to Old Maid Simulator! ";
        h.display(str);
   deck = new Deck();
   deck.shuffle();
}

@Override
public void runGame(){
   playRound();

}

/**
   * Lets the user play one game of HighLow, and returns the
   * user's score in that game.  The score is the number of
   * correct guesses that the user makes.
   */
 @Override
 public void playRound() {
     Card newCard;
```

```
int players, card, human = 0, nextplayer, sum = 0;
players = h.getPlayers("How many people will be playing today?");
Hand[] group = new Hand[players];

for(int i = 0; i < group.length; i++) {
	group[i] = new Hand(); //Instantiate
}

for(int i = 0; i < deck.cardCt; i++) {
	for(int j = 0; i < deck.cardCt; j++) {
		newCard = deck.dealCard();
		group[j].addCard(newCard);

		if(j == (group.length - 1))
			j=-1;
		i++;
	}
}

for(int i = 0; i < group.length; i++) {//The Initial hand
	h.display("\n>Player" + (i+1) +"'s Initial Hand");
	for(int j = 0; j < group[i].getCardCount(); j++) {
		System.out.println(group[i].getCard(j));
	}
}


for(int i = 0; i < group.length; i++) {//Sort the Cards and remove initial Pairs
	h.display("\n>Player" + (i+1) +"'s Initial Pairs");
```

```
        group[i].sortByValue();
        group[i].removePairs(group[i]);
}



while(true) {
        for(int i = 0; i < group.length; i++) {//The Initial hand
                nextplayer = i+1;

                if(nextplayer >= group.length )
                        nextplayer = 0;

                while(group[nextplayer].getCardCount() == 0){
                        nextplayer++;
                        if(nextplayer >= group.length )
                                nextplayer = 0;
                }

                while(group[i].getCardCount() == 0){
                        i++;
                        if(i >= group.length )
                                i = 0;
                }

                h.display("\n>Player" + (i + 1) +"'s Turn");

                if(i == human) {
                        card = h.grabCard(1, group[nextplayer].getCardCount());
                }
```

```
                else {
                        Random r = new Random();
                        card = r.nextInt(group[nextplayer].getCardCount());
                }

                h.display("\n>Player" + (i + 1) +" got the " + group[nextplayer].getCard(card));
                group[i].addCard(group[nextplayer].getCard(card));
                group[nextplayer].removeCard(card);
                group[i].sortByValue();
                group[i].removePairs(group[i]);

                int player = 0;

                for(int j = 0; j < group.length; j++) {
                        int num = group[j].getCardCount();
                        sum += num;
                        if(num > 0)
                                player = i + 1;

                }

                if(sum == 1) {
                        h.display("Player" + player + " had the Old Maid, they lose!");
                        System.exit(1);
                }
                sum = 0;
        }

}
```

```
        } //end playRound()


}


public Deck(boolean includeJokers) {
        if (includeJokers)
            deck = new Card[54];
        else
            deck = new Card[51];
        for ( int suit = 0; suit <= 3; suit++ ) {
            for ( int value = 1; value <= 13; value++ ) {

             if(suit == 1 && value == 12) {
             }
             else {
                    deck[cardCt] = new Card(value,suit);
                    cardCt++;
             }
            }
        }
        if (includeJokers) {
            deck[52] = new Card(0, Card.JOKER);
            deck[53] = new Card(14, Card.JOKER);
        }
        cardsUsed = 0;
    }
```

Nach Streveler UML