Name _____ Period _____

Card Dealer

Your Tasks (Mark these off as you go)

- ☐ Define key vocabulary
- ☐ Review the Card class
- ☐ Review the DeckOfCards class
- ☐ Write the shuffleCards method in the DeckOfCards class
- ☐ Deal a shuffled array of cardsComplete
- ☐ Complete the CardDealer class
- ☐ Receive credit for this lab guide

☐ Define key vocabulary

Static method

☐ Review the card class

The Card class models a playing card from a standard 52 card deck. Creating a Card requires three parameters: the face value of the card (1 thru 10, jack, king, queen, or ace), a suite name (spades, diamonds, hearts, or clubs), and a card value (1 thru 13, where ace equals 1). Once the card is created, there are three getters methods which allow us to access these variables.

Below illustrates how to use the Card class to create a card,

```
Card card1 = new Card("ace", "spades", 1);
```

Notice, the methods in the Card class are nonstatic. This is because we will need to create 52 unique cards.

Card Class

```
/**
 * Models a Card from a standard playing deck
 * @author Pluska
 */
public class Card {

    private String faceValue, suite;
    private int value;

    /**
     * Card Constructor
     * @param fv face value of the card, (e.g., king, one, two)
     * @param s suite of the card
```

```
     * @param v value of the card (1-13)
     */
    public Card(String fv, String s, int v){
        suite = s;
        faceValue = fv;
        value = v;
    }

    /**
     * Gets the value of the card
     * @return value of card (1-13)
     */
    public int getValue(){
        return value;
    }

    /**
     * Gets the face value of the card
     * needed to display the card on the GUI
     * @return properly formatted face value
     */
    public String getFaceValue(){

        switch(faceValue){
            case "one":
                return "1";
            case "two":
                return "2";
            case "three":
                return "3";
            case "four":
                return "4";
            case "five":
                return "5";
            case "six":
                return "6";
            case "seven":
                return "7";
            case "eight":
                return "8";
            case "nine":
                return "9";
            case "ten":
                return "10";
            case "jack":
                return "J";
            case "queen":
                return "Q";
            case "king":
```

```
                return "K";
            case "ace":
                return "A";
        }
        return "";
    }

    /**
     * Gets the name of the suite
     * needed to display the card on the GUI
     * @return the name with the first letter capatilized
     */
    public String getSuite(){
        String tempSuiteArray[] = suite.split(" ");
        String firstLetter = tempSuiteArray[2].substring(0,1).toUpperCase();
        String everythingElse = tempSuiteArray[2].substring(1);
        return firstLetter+everythingElse;
    }

    public String toString(){
        return faceValue+suite;
    }
}
```

Refer to the card class above. Write code that could be used to create the following "hand" of cards, Ace of spades, King of hearts, Queen of hearts, Jack of clubs, 7 of diamonds

Refer to the Card objects you created above, write code that could be used to print the value and suit of the Ace of Spades Card object.

□ Review the DeckOfCards class

The DeckOfCards class models a DeckOfCards. Notice, the DeckOfCards class does not have a constructor explicitly defined and all the methods are static. This is because when we play a game with cards, we typically only want one deck.

Recall that static methods can be accessed without creating an object of that class. For example, the following call from the main method would build a sorted DeckOfCards,

```
DeckOfCards.buildDeck();
```

Once the deck is build, we can use the dealCards method to deal a specified number of cards,

```
DeckOfCards.dealCards(5);
```

DeckOfCards class

```
/**
 * Models a standard Deck of Cards
 * @author Pluska
 */
public class DeckOfCards {

    private static Card cards[];
    private static Card cardDeal[];
    private static final int DECKSIZE = 52;
    private static int nextCardIndex = 0;
    private static int cardsLeft = 52;

    private static String[] suiteNames = {
        //The symbols for the suites do
        //not display properly in the windows console
        " of spades " + '\u2660',
        " of diamonds " + '\u2666',
        " of clubs " + '\u2663',
        " of hearts " + '\u2764'

    };
    private static String[] values = {
        "ace", //0
        "two", //1
        "three", //2
        "four", //3
        "five", //4
        "six", //5
        "seven", //6
        "eight", //7
        "nine", //8
        "ten", //9
        "jack", //10
        "queen", //11
```

```
        "king"//12
    };

    /**
     * DO NOT EDIT
     * Creates a sorted deck of 52 cards
     */
    public static void buildDeck(){

        cards = new Card[DECKSIZE];//creates an array of cards
        int cardValueIndex = 0;

        for(int s = 0; s < suiteNames.length; s++){
            for(int v = 0; v < values.length; v++){
                //creates a card object at each index of cards
                cards[cardValueIndex] = new Card(values[v], suiteNames[s], v);
            cardValueIndex++;
            }
        }
    }

    /**
     * DO NOT EDIT
     * Returns the next Card in the deck
     * @return
     */
    public static Card nextCard(){
        nextCardIndex++;
        cardsLeft--;
        return cards[nextCardIndex-1];
    }

    /**
     * DO NOT EDIT
     * Deals a set of cards of a specified length
     * @param dealSize
     * @return
     */
    public static Card[] dealCards(int dealSize){
        cardDeal = new Card[dealSize];
        for(int c = 0; c < dealSize;c++){
            cardDeal[c] = nextCard();
        }
       cardsLeft -= dealSize;
       return cardDeal;
    }

    /**
     * DO NOT EDIT
```

```
     * @return cards left in the deck
     */
    public static int getCardsLeft(){
        return cardsLeft;
    }
}
```

☐ Write the shuffleCards method in the DeckOfCards class

Notice, the cards that are dealt in the DeckOfCards class are sorted. Playing cards requires that we have a shuffled deck. We can shuffle our deck by swapping random cards in the deck over and over again. In the spaces below, you will write code that will allow you to create a shuffled deck of cards.

The signature for the swapCards method is as follows,

```
public static void swapCards(int i1, int i2)
```

i1 and i2 represent the indices of Card objects in the cards array. Write the swapCards method below which swaps two Cards in the cards array.

The signature for the shuffleCards method is as follows,

```
public static void shuffleCards()
```

Write a loop to shuffle the cards. Inside this loop you should create two random numbers which represent the indices of two cards you want to swap. Then call swap. Your loop can be repeated as many times as you like to ensure the cards are adequately shuffled.

□ Deal a shuffled array of cards

The purpose of the CardDealer class is to do the things that CardDealers typically do, shuffle cards, deal cards, etc. To do this, CardDealer will need access to the methods in the DeckOfCards class.

The card dealer class will contain the main method for your program. Inside this method you will be able to build a shuffled DeckOfCards using the following calls.

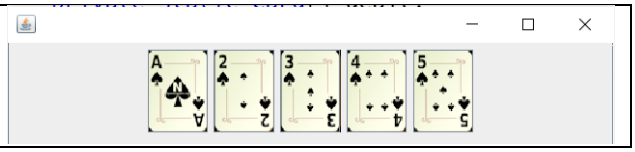
```
DeckOfCards.buildDeck();
DeckOfCards.shuffleCards();
```

Likewise, you deal a set of cards. Notice, that the dealCards method in the DeckOfCards class returns an array of Cards. In the code below we have assigned the array returned to this method to a Card array called dealt.

```
dealt = DeckOfCards.dealCards(5);
```

Using the GUI built in to this project allows to view the dealt cards,

```
displayCards();
```



As cards are dealt, we have the potential of running out of cards and hence going out of bounds. We therefore need to check for this.

Add an if statement to the line of code below, to check if there are enough cards in the deck to deal a complete hand. Call the getCardsLeft method in the DeckOfCards class to check for this. If there are enough cards to complete the deal, you may deal the cards, otherwise, you must build a new DeckOfCards, shuffle, and re-deal.

```
dealt = DeckOfCards.dealCards(5);
```

□ Complete the CardDealer class

We can write additional methods in the CardDealer class to add functionality to our program. Because we will be calling these methods from a static context, the methods we write in the CardDealer class will also need to be static.

Consider the line of code below, which creates an array of cards and assigns them to a Card array data type called dealt,

```
dealt = DeckOfCards.dealCards(5);
```

The card at a given index can be accessed as follows,

```
Card index0 = dealt[0];
```

Likewise, the value of the card expressed as a number (1-13) can be accessed by calling the getValue method in the Card class,

```
int cardVal = index0.getValue();
```

In the space below, write the getHighestCard method. The getHighestCard method should find and return the card with the highest value in a given array of dealt cards. The getHighestCard method has the following signature,

```
public static Card getHighestCard()
```

In the space below, write the getLowestCard method. The getLowestCard method should find and return the card with the lowest value in a given array of dealt cards. The getLowestCard method has the following signature,

```
public static Card getLowestCard()
```

Now that we know how to get the value of each Card, write the sumDeal method, which sums and returns the value of a given array of dealt cards. The sumDeal method has the following signature,

```
public static int sumDeal()
```

Write a method that could be used to count and return the number of each suite in a given array of dealt cards. The suite of a card can be accessed using the getSuite method in the Card class. Below, is an example of how to get the suite of the card at index 0 in the dealt array.

```
String suitOfCard = dealt[0].getSuite();
```

The number of each suite should be stored in the suite array,

```
private static int suiteArray[] = new int[4];
```

Where index 0 can represent clubs, index 1 can represent diamonds, etc.

The getSuites method should have the following signature,

```
public static int[] getSuites()
```

Write a method that could be used to count and return the number of each card value in a given array of dealt cards. The value of a card can be accessed using the getValue method in the Card class. Below, is an example of how to get the value of the card at index 0 in the dealt array.

```
int valueOfCard = dealt[0].getValue();
```

The number of each value should be stored in the ValuesArray,

```
private static int valuesArray[] = new int[14];
```

Where index 1 can represent the number of cards with a value of 1, index 2 can represent the number of cards with a value of 2, etc.

The getValues method should have the following signature,

```
public static int[] getValues()
```

□ Receive credit for this lab guide

Submit this portion of the lab to Pluska to receive credit for the lab guide. Once received, your completed code challenges will also be graded and will count towards your final lab grade.