```java
/*
 * Blair Hagen
 * Lab 4
 * 4-23-2016
 *
 * As a student at Union College, I am part of a community that values
intellectual effort, curiosity and discovery. I understand that in order to
truly claim my educational and academic achievements, I am obligated to act
with academic integrity. Therefore, I affirm that I will carry out my
academic endeavors with full academic honesty, and I rely on my fellow
students to do the same.
 *
 * Working with vector data types and understanding data hiding
 *
 */
import java.util.Vector;
import java.util.Random;

/**
 * Simulates a deck of playing cards.  This deck has a few more methods than
 * the one you built for Project 2.
 */
public class Deck {
      private final int NUMBER_OF_CARDS=52;
      private final int NUMBER_OF_SUITS=4;
      private final int CARDS_IN_SUIT=13;

    private Vector<Card> theCards;
    private boolean shuffled;

    /**
     * Makes a new ordered deck of playing cards
     */
    public Deck()
    {
        theCards = new Vector<Card>(NUMBER_OF_CARDS);
        shuffled=false;

        int suitIndex = 0;
        int cardIndex = 1;
        for (int i = 0; i < NUMBER_OF_CARDS; i++)
        {
            Card newCard = new Card(cardIndex, suitIndex);
            theCards.add(newCard);
            cardIndex++;

            if (cardIndex == 14)
            {
                  cardIndex = 1;
                  suitIndex++;
            }
        }
    }

    /**
     * Deals out next card in deck; returns null if no cards left
     *
 * @return next card in deck or null if deck empty
 */
public Card deal()
{
  Card returnCard;
    if (theCards.size() == 0)
    {
        returnCard = null;
    }
    else if (shuffled)
    {
        Random rand = new Random();
        int randomIndex = rand.nextInt(theCards.size());
        returnCard = theCards.elementAt(randomIndex);
        theCards.remove(randomIndex);
    }
    else
    {
        returnCard = theCards.firstElement();
        theCards.remove(0);
    }

    return(returnCard);
}

/** Tells if deck has any cards left in it
 *
 * @return true if Deck empty; else false
 */
public boolean isEmpty()
{
  if (theCards.size() == 0)
  {
        return(true);
  }
  else
  {
        return(false);
  }
}

/**
 * Shuffles the cards
 */
public void shuffle()
{
    shuffled = true;
}

/** Returns number of undealt cards left in the deck
 *
 * @return number of undealt cards in the deck
 */
public int size()
{
  return(theCards.size());
}
```

```
    /**
     * Reset the deck by gathering up all dealt cards.
     * Postcondition: Deck contains all cards and is shuffled
     */
    public void gather()
    {
      int valueInt;

      for (String valueString:Card.values)
      {
            if (valueString.equals("Ace"))
            {
                  valueInt = 1;
            }
            else if (valueString.equals("Jack"))
            {
                  valueInt = 11;
            }
            else if (valueString.equals("Queen"))
            {
                  valueInt = 12;
            }
            else if (valueString.equals("King"))
            {
                  valueInt = 13;
            }
            else
            {
                  valueInt = Integer.parseInt(valueString);
            }

            for (String suitString:Card.suits)
            {
                  Card searchCard = new Card(valueInt, suitString);
                  if (theCards.indexOf(searchCard) == -1)
                  {
                        theCards.add(searchCard);
                  }
            }
      }
      shuffled = true;
    }

    /**
     *  DEBUGGING METHOD: prints out stats of deck, that is, the
<i>undealt</i> cards.
     *  Prints the remaining number of cards of each suit and of each value.
     */
    public void printStats()
    {
        int Hcount=0;
        int Dcount=0;
        int Scount=0;
        int Ccount=0;
        int[] values = new int[CARDS_IN_SUIT];
        int size=theCards.size();
        for (int i=0; i<size; i++)
        {
            int val = theCards.elementAt(i).getValue();
            String suit = theCards.elementAt(i).getSuit();
            if (suit.equals("clubs"))
                Ccount++;
            else if (suit.equals("diamonds"))
                  Dcount++;
            else if (suit.equals("spades"))
                  Scount++;
            else if (suit.equals("hearts"))
                Hcount++;
            values[val-1]++;  // deck values run from 1-13 so need to
subtract 1
        }
        System.out.println("***PRINTING DECK STATS***");
        System.out.println("# clubs: " + Ccount);
        System.out.println("# diamonds: " + Dcount);
        System.out.println("# hearts: " + Hcount);
        System.out.println("# spades: " + Scount);

        System.out.print("Card:\t");
        for (int j=0; j<values.length; j++) {
            System.out.print(Card.values[j]+"\t");
        }
        System.out.println();
        System.out.print("Qty:\t");
        for (int j=0; j<values.length; j++) {
            System.out.print(values[j] + "\t");
        }
        System.out.println("\n");
    }
}
```

```
import java.util.Vector;

/** Driver for Lab 4 **/
public class Client {

      public static void main(String[] args) {
//          sandbox();
//          inOrder();
//          shuffledOrder();
//          dealThenShuffle();
            gatherTest();
      }

      /**
       * Just a play area for you to try out the Card class.
       */
      public static void sandbox() {
            Card oneCard = new Card(12, "diamonds");
            System.out.println(oneCard.getValue());
            System.out.println(oneCard.getSuit());
            System.out.println(oneCard.toString());

            Card twoCard = new Card(11, 3);
            System.out.println(twoCard.toString());
            System.out.println("--------------------");

          Vector<Card> testCard;
            testCard = new Vector<Card>(10);
            testCard.add(oneCard);
            testCard.add(twoCard);
            System.out.println(testCard.get(0).toString());
            System.out.println(testCard.get(1).toString());

      }

      /**
       * DECK TEST: Constructs a deck and prints it (should be in order).
       * This tests the constructor.
       */
      public static void inOrder(){
            System.out.println("IN ORDER TEST");
            Deck deck1 = new Deck();
            deck1.printStats();
            dealAndPrint(deck1);
      }

      /**
       * DECK TEST: Constructs a deck, shuffles, and prints it.
       * This tests the <code>shuffle</code> method to see if it shuffles all
cards.
       */
      public static void shuffledOrder(){
            System.out.println("SHUFFLED ORDER TEST");
            Deck deck2 = new Deck();
            deck2.printStats();
            deck2.shuffle();
            dealAndPrint(deck2);
      }

      /**
       * DECK TEST: Deals first 3 (ordered) cards, shuffles, then prints the
rest.
       * This tests the <code>shuffle</code> method to see if it shuffles
remaining cards.
       */
      public static void dealThenShuffle(){
            System.out.println("DEAL IN SORTED ORDER, THEN SHUFFLE THE
REST");
            Deck deck3 = new Deck();
            System.out.println(deck3.deal());
            System.out.println(deck3.deal());
            System.out.println(deck3.deal());
            deck3.shuffle();
            deck3.printStats();
```

```
		dealAndPrint(deck3);
	}

	/**
	 * DECK TEST: Deals an ordered deck, prints number left in deck (should
be zero),
	 * gathers cards, prints number left in deck (should be all), shuffles,
and deals all.
	 * This tests the <code>gather</code> and <code>size</code> methods.
	 */
	public static void gatherTest(){
		System.out.println("GATHER METHOD TEST");
		Deck deck4 = new Deck();
		dealAndPrint(deck4);
		System.out.println("Before gathering, deck has " + deck4.size() +
" cards.");
		deck4.printStats();
		deck4.gather();
		System.out.println("After gathering, deck now has " +
deck4.size() + " cards.");
		deck4.printStats();
		deck4.shuffle();
		dealAndPrint(deck4);
	}

	/**
	 * Use this method to help you debug.  It will deal out all cards in
the deck.
	 * @param theDeck the deck to deal
	 */
	public static void dealAndPrint(Deck theDeck){
		System.out.println("dealing all cards:");
		System.out.println("-------------------");
		if (theDeck.isEmpty()){
			System.out.println("### No cards in deck! ###");
		}
		while (!theDeck.isEmpty()){
			System.out.println(theDeck.deal());
		}
		System.out.println();
	}

}
```