```
1 import java.util.ArrayList;
 2 import java.util.Random;
 3 import java.util.concurrent.ThreadLocalRandom;
 4
 5 /**
   * Simulates a deck of 52 playing cards.
 7
   */
 8 public class Deck {
       private static final int NUMBER_OF_CARDS=52;
       private static final int NUMBER_OF_SUITS=4;
10
11
       private static final int CARDS_IN_SUIT=13;
12
       private static final int FIRSTCARDINDEXINDECK = 0
13
14
       private ArrayList<Card> theCards;
15
       private boolean shuffled;
16
17
       /**
18
        * Constructs a new ordered deck of playing cards
19
        */
20
21
       public Deck()
22
       {
23
           theCards = new ArrayList<Card>(
   NUMBER_OF_CARDS);
24
           shuffled= false;
25
           addCardsToDeck();
26
27
       }
28
29
30
       /**
31
        * Add a standard 52 cards to a deck
32
        */
33
       private void addCardsToDeck() {
34
           for (String suit: Card.SUITS){
35
               for (int rank = Card.MINRANK; rank <=</pre>
   Card.MAXRANK; rank++){
                   theCards.add(new Card(rank, suit));
36
37
               }
           }
38
```

```
39
40
41
42
       /**
43
        * Deals out next card in deck; returns null if
   no cards left
44
45
        * <u>@return</u> next card in deck or null if deck
   empty
46
        */
47
       public Card deal() {
48
           Card cardToDeal;
49
           if (!shuffled && !isEmpty()) {
50
51
                cardToDeal = theCards.get(0);
52
                theCards.remove(cardToDeal);
53
                return cardToDeal;
           }
54
55
56
           else if (shuffled && !isEmpty())
                int randomCardIndex = ThreadLocalRandom.
57
   current().nextInt(FIRSTCARDINDEXINDECK, size());
58
                cardToDeal = theCards.get(randomCardIndex
   );
                theCards.remove(cardToDeal);
59
60
                return cardToDeal;
61
           }
62
           else { return null; }
63
       }
64
65
66
       /** determines if deck has any cards left in it
67
68
        * @return true if Deck empty; else false
69
       public boolean isEmpty() {
70
71
           return size() == 0;
72
       }
73
74
       /**
75
        * Shuffles the cards
```

```
76
         */
 77
        public void shuffle()
 78
        {
 79
             shuffled = true;
 80
        }
 81
 82
        /** Returns number of undealt cards left in the
    deck
 83
 84
         * <u>@return</u> number of undealt cards in the deck
 85
         */
 86
        public int size()
 87
 88
             return theCards.size();
 89
        }
 90
 91
        /**
 92
         * Reset the deck by gathering up all dealt
    cards.
 93
         * Postcondition: Deck contains all cards and is
     NOT shuffled
 94
         */
 95
        public void qather()
 96
        {
 97
             theCards.clear();
 98
             addCardsToDeck();
 99
             shuffled = false;
100
101
        }
102
103
        /**
            DEBUGGING METHOD: prints out stats of the
104
    given list of cards, that is, what was dealt.
105
         * Prints the remaining number of cards of each
     suit and of each rank.
106
         *
            <u>Oparam</u> cardList list of cards that are (were
107
    ) in the deck
         * <u>@hidden</u>
108
109
         */
110
        public void printStats(ArrayList<Card> cardList)
```

```
111
        {
112
            int Hcount=0;
113
            int Dcount=0;
114
            int Scount=0;
115
            int Ccount=0;
116
            int[] ranks = new int[CARDS_IN_SUIT];
117
            int size=cardList.size();
            for (int i=0; i<size; i++)</pre>
118
119
            {
120
                 int val = cardList.get(i).getRank();
                 String suit = cardList.get(i).getSuit();
121
                if (suit.equals("clubs"))
122
123
                     Ccount++;
                else if (suit.equals("diamonds"))
124
125
                     Dcount++;
                else if (suit.equals("spades"))
126
127
                     Scount++;
                else if (suit.equals("hearts"))
128
129
                     Hcount++;
                ranks[val-2]++; // deck RANKS run from
130
    2-14 so need to subtract 2
131
132
            System.out.println("***PRINTING DECK STATS
    ***");
133
            System.out.println("# clubs: " + Ccount);
            System.out.println("# diamonds: " + Dcount);
134
            System.out.println("# hearts: " + Hcount);
135
            System.out.println("# spades: " + Scount);
136
137
            System.out.print("Card:\t");
138
            for (int j = 2; j < Card.RANKS.length; j++) {</pre>
139
            System.out.print(Card.RANKS[j]+"\t");
140
            }
141
142
            System.out.println();
            System.out.print("Qty:\t");
143
            for (int j=0; j<ranks.length; j++) {</pre>
144
            System.out.print(ranks[j] + "\t");
145
            if (j>8) { // indices 9-12 are Jack thru
146
    Ace
147
                 System.out.print("\t");
148
                 }
```

```
File - /home/dateraon/Desktop/CSC-120-Git/CSC-120/Lab08 Neil Daterao/src/Deck.java
149
              System.out.println("\n");
150
151
          }
152
153
          /**
              DEBUGGING METHOD: prints out stats of this
154
           *
     Deck object
155
           * Prints the remaining number of cards of each
      suit and of each rank.
156
           *
           * <u>@hidden</u>
157
158
           */
          public void printStats() {
159
160
              printStats(theCards);
          }
161
162
163
164 }
165
```