FALL 2016: **COP 3530** Data Structures

[Programming Assignment 1; Due September 14 before start of class.]

Problem Description

Imagine that you have been given a regular deck of 52 cards. At the start, the deck of cards is in **perfect increasing order**, i.e., the deck when placed face down has the four Aces at the top followed by the four twos, four threes, . . ., four tens, four jacks, four queens, and four kings. Within the set of four Aces, the cards are ordered starting from the `Club Ace` at the top followed by the `Diamond Ace`, `Heart Ace`, and the `Spade Ace`. The same ordering of the suites occurs in the twos, threes and so on. The objective is to implement a **perfect shuffle**.

Implement classes called `aCard` and `deckOfCards`, as shown below. Each card in the deck can be "coded" as an integer `cardCode`. For example, this integer code could refer to the position of that card in the initial deck. The method `toString` could take care of converting from the integer code to the string representing the card. The class `deckOfCards` should extend a `ArrayList` or `LinkedList` of `aCard` and should be initialized to contain the deck in perfect increasing order. Also, define an interface called `Deck` and have the class `deckOfCards` implement `Deck`.

Define a **perfect shuffle** as follows. The deck of cards is placed face down on a table. It is then separated into two equal half piles, and then the two half piles are "perfectly interleaved" (just like the way the professional card dealers shuffle at a casino). In other words any two cards that are next to each other in one of the half piles, will be separated by exactly one card from the other pile after the perfect shuffle. Care is taken so that the card that was on top of the deck before the shuffle remains on top after the shuffle (similarly, the card that was at the bottom of the deck before the shuffle remains at the bottom after the shuffle). Note that the method, as defined above, has no randomness in the result. A call to the method `perfectShuffle` should achieve the above task.

There is one special card in the deck – the *Jack of Spades*. The method `findSpecial` should report the location of the special card (i.e., its position from the top of the deck, where the top card is said to have location 0 and the bottom card 51).

Next imagine that a dealer deals 4 hands for a card game, i.e., the dealer deals one card per player iteratively until all the cards in the deck are distributed. So the first player (player number 0) would get cards from locations $0, 4, 8, 12, \ldots, 48$, while the second player (player number 1) gets cards from locations $1, 5, 9, 13, \ldots, 49$, and so on. The method `findMax` should report the "highest card" in a player's hand. In order to determine the highest card, the cards in the deck in increasing order (the order used in the game "Bridge") are as follows: `Two Clubs`, `Two Diamonds`, `Two Hearts`, `Two Spades`, `Three Clubs`, and so on, upto `King Clubs`, `King Diamonds`, `King Hearts`, `King Spades`, finally followed by `Club Ace`, `Diamond Ace`, `Heart Ace`, `Spade Ace`. Note that `findMax` has the total number of players as a parameter and the player number whose hand it analyzes to find the highest card.

Finally, write a main program that creates a new deck of cards, shuffles the deck N times, and after each shuffle reports both the location of the special card, as well as the highest card in the hand of each player number (assuming that there are M players and the cards are dealt after that shuffle). You will get a data file called `h1Data.txt` with the two integers N and M.

What to Submit

Do not print out the contents of the decks after each shuffle, although you may want to add a private method to do so for debugging purposes. Make sure to have a good comments section and run your code through Javadoc. Use runtime exceptions to check parameters of `findMax`. Submit on the moodle class website. At the head of the program you should have your name and class information. Your program is due before the start of the class on the date mentioned.

For every programming assignment, print out a statement at the top of the program stating that "the program is your work and you did not acquire it or part of it from elsewhere". It may be possible to submit this statement via Moodle too.

Specification you need to use

```
public interface Deck
{
    public void perfectShuffle();
    public int locateSpecial();
    public Object findMax( int numberPlayers, int playerNumber )
}

class aCard implements Comparable<aCard>
{    /* Constructor(s) needed */
     public String toString()        { /* Implementation needed */ }
     public int compareTo(aCard c) { /* Implementation needed */ }
     private int cardCode;     /* private data field */
}

class deckOfCards extends ArrayList<aCard> implements Deck
{   /* Constructor(s) needed */
    /* Implementations of perfectShuffle, locateSpecial, and findMax needed */
}
```

Extra Credit

For extra credit you may try your hand at any of the various problems listed below. Make sure your documentation states clearly which of these problems you have correctly implemented. It is your responsibility to prove to me that your implementations of these parts work correctly.

1. Add a method to generate a "random shuffle", i.e., a random permutation. Your documentation should explain clearly the algorithm you are using to generate such a shuffled deck.
2. Write a program to count the number of "random shuffles" you need before the "Heart Queen" is at the top of the deck.
3. Write a program to count the number of "random shuffles" before player number 1 gets at least 6 cards from the Diamonds suit.