PLAYING CARDS

Yesterday, you learnt from a mysterious man that there is a lot of money in the business of virtual casinos and that sounded like a job exactly suitable for a CS101 student which is good at designing gambling games (especially ones with dice). Since the most important games in your casino would be the playing card games, you decided to put extra effort at writing a good playing card class.

You numbered some terms about playing cards to be able to write numerical algorithms for them:

• A	– 1
• J	– 11
• Q	– 12
• K	– 13

• Clubs (♣)	– 0
• Diamonds (♦)	– 1
• Spades (♠)	– 2
• Hearths (♥)	– 3

Then took notes about implementation details of planned classes.

CARD CLASS

This class will hold the information of a card: its value and its suit (clubs, diamonds …) as both integers. Here is the list of methods to be implemented:

1) A constructor which takes two integers for both properties.
2) A smallerThan() method which returns true if a card is smaller than another card.

In your casino, the card with the smaller suit number is the smaller one, if the suit numbers are equal then you compare their values. After some trial games with your friends, you noticed this order between cards to be more comfortable to hold at hands.

For example: King of Clubs is smaller than Nine of Hearths
Two of Spades is smaller than Queen of Spades

3) A toString() method which will return results in the form of "Ten of Diamonds"
4) An equals() method which will check if two cards are actually the same cards.

DECK CLASS

This class will hold any number of non-repeating cards in an ArrayList of Cards. Here is the list of methods to be implemented:

1) A default constructor which will initialize the deck as an empty deck.
2) An addRandom() method which will add a random card to the end of the deck. (Be careful not to add any repeating cards!)
3) A sort() method which will sort the deck according to the smallerThan() function of cards.

For example: This is a sorted deck: ♣1 ♣Q ♠10 ♥A ♥3

4) A shuffle() method which will shuffle the deck.

5) A cut() method which will cut the deck half at a random point and reassemble it in reverse order.

For example: ♣1 ♣Q ♥3 / ♥A ♠10 (cutting from the third card) ♥A ♠10 / ♣1 ♣Q ♥3

6) A toString() method which returns all cards in the deck, line by line.