## Introduction

Well, you created the Card class, guess what's next? You guessed it, the Deck class.

## Deck Class

What does a real-world deck look like? Something like a stack of cards. Let's make things simple by creating a deck of 52 cards with no jokers. But we are getting a bit ahead of ourselves. Let's talk about class attributes and methods.



## Attributes

We will need to represent 52 cards and treat each card uniformly. Hey, that sounds like an array.

The Deck class will need a constructor. The constructor should not take any parameters. if it did, what parameters could you pass? The 52 individual cards? No, let's make the constructor create the card objects and save a reference to each card in an array that looks like this:

private Card [] cards; // must allocate space for 52 cards and fill each   
 //element with a Card object

The class will need another data member to keep track of which card to deal next, let’s call it the topCard and it's just an index into the cards array.

private int topCard; // range: 0 .. 51

## Methods

Well what should our Deck class do? How about shuffling its cards and dealing out some cards? Below is the signature of the shuffle method.

public void shuffle()

The shuffle method takes no parameters and returns no value (void). It only has the side-effect of randomizing the elements in the cards array.

Here is pseudo code for a shuffle algorithm.

-- To shuffle an array *a* of *n* elements (indices 0..n-1):

**for** *i* **from** 0 **to** *n*−2 **do**

*j* ← random integer such that 0 ≤ *j* < *n*

swap *a*[*i*] and *a*[j]

You will want to use this technique to generate a random integer in the proper range.

j = (int) (Math.random() \* 52);

Remember that to swap two elements in an array (e.g. a[i] and a[j] you will need a temporary variable to hold the value of the first element that is assigned to.

Let's think about the deal() method. It is local to the class so it can directly reference the cards data member. What does it need to know? Answer, how many cards to deal which we will call count. And what should it return to the caller? An array of count cards. Here is the signature of deal().

public Card[] deal(int count)

Now deal must avoid dealing the same cards as it did to previous callers (players). It can do this by using the topCard variable to keep track of the "top card" index into the cards array. Here is pseudo-code for deal() method.

-- deal count cards

**if** topCard + count > 51 **then exception**

**allocate space for local array of count cards called** deal

**for** *i* **from** 0 **to** *count-1* **do**

*deal[i]* ← cards[topCard + i]

topCard ← topCard + count

**return** deal

For this class there is no need for getters and setters. It doesn't make sense to compare two decks (how would you do it?). It makes some sense to have a toString() method but that method would have to return a very long string indeed. So, let's not do that.

## MainClass

Let's use the class (e.g. MainClass.java) containing the main() method to test our Deck class directly and the Card class indirectly.

First add a static method to show a "hand" that is dealt from the deck.

private static void show(Card[] hand) {

int count = hand.length;

for (int i = 0; i < count; i++) {

System.out.print(hand[i].toString() + " ");

}

System.out.println();

}

Now add these lines to your main() method to see how the Deck's methods are doing.

Deck deck = new Deck();

Card[] hand1 = deck.deal(13);

Card[] hand2 = deck.deal(13);

Card[] hand3 = deck.deal(13);

Card[] hand4 = deck.deal(13);

System.out.println("A deck:");

show(hand1);

show(hand2);

show(hand3);

show(hand4);

Do you see a full deck of 52 cards? If the answer is "no" go back and fix your Deck constructor. Does each hand reflect a full suit (e.g. CLUBS)?

Now let's create a new deck (why must you create a new deck?) and shuffle the cards before dealing them out.

Deck deck1 = new Deck();

deck1.shuffle();

Card[] hand5 = deck1.deal(13);

Card[] hand6 = deck1.deal(13);

Card[] hand7 = deck1.deal(13);

Card[] hand8 = deck1.deal(13);

System.out.println("A shuffled deck:");

show(hand5);

show(hand6);

show(hand7);

show(hand8);

Are the cards in random order? Are there any duplicated cards? Are there any missing cards? If the answer is "no" to the first question or "yes" to either the second or third question go back and fix the shuffle method.

## Assignment

Get your Card and Deck classes working so that the test code in main() produces the correct results. Submit your Deck.java. In a comment write your reflections on "what I learned" when doing this assignment.