Elevens Lab - Haas

This is the third and final official AP Computer Science Lab activity.
The original lab was 42 pages!!! ***"Yes that's right - I cut it down to 4!!!"***

First – ***download*** and ***extract all*** the Elevens Lab Code and ElevensGame.jar from:
https://drive.google.com/drive/folders/1OsjR8G1Pv4B5OjLIxHmje2praDQ4FuQb?usp=sharing

You will complete several methods to play the card game Elevens. You don't need to understand how the whole thing works, just the parts that you need to complete.

Check out the finished game by running the ElevensGame Executable Jar File.

- Click on **two** cards that sum up to 11 then click Replace.
- An ace is worth 1, and suits are ignored.
- Face cards can only be removed as a triplet consisting of a J, a Q, and a K.
- Cards are dealt from the deck if possible to replace the cards just removed.

Activity 1: The Card Class
Open the Elevens Lab Code in BlueJ

1. Complete the implementation of the following methods in the **`Card`** class.
   - the three parameter **constructor**
   - accessor methods `rank(), suit(),` and `pointvalue()`
   - method `matches(Card otherCard)` to test equality between two card objects
   - `toString()` - that returns the rank, suit, and point value of the card:
     `rank` `of` *`suit`* `(point value =` *`pointValue`*`)`
     for example: `ace of hearts (point value = 1)`

2. Test your code by running the main method in the **`Card`** class and check your output.

rank: ace suit: clubs pointValue: 1
toString: ace of clubs (point value = 1)

rank: ace suit: clubs pointValue: 1
toString: ace of clubs (point value = 1)

rank: 6 suit: hearts pointValue: 6
toString: 6 of hearts (point value = 6)

aceClubs1.matches(aceClubs2) true
aceClubs1.matches(sixHearts) false

Activity 2: The Deck Class

1. Complete the implementation of the **Deck** class by coding each of the following:

- **Deck constructor** - The constructor receives three arrays as parameters. The arrays contain the ranks, suits, and point values for each card in the deck.

  a) The constructor sets ArrayList `cards` to a new ArrayList, and then creates the specified cards and adds them to the list.

  For example: if `ranks={"jack","queen","king"};`
  `suits={"clubs","hearts"}; pointValues={11,12,13};`

  The following cards would be added to the ArrayList:

  ```
  [jack,clubs,11],[jack,hearts,11]
  [queen,clubs,12],[queen,hearts,12],
  [king,clubs,13], [king,hearts,13],
  ```

  b) Set parameter `size` to the size of `cards`, which in this example is 6.

  c) Shuffle the deck by calling the `shuffle` method. Note that you will not be implementing the shuffle method until Activity 3.

- **isEmpty** - This method should return true when the size of the deck is 0; false otherwise.

- **size** - This method returns the number of cards in the deck that are left to be dealt.

2. Test your code by running the main method in the **`Deck`** class and check your output.

shuffle is not yet implemented
size = 6
jack of clubs (point value = 11)
jack of hearts (point value = 11)
queen of clubs (point value = 12)
queen of hearts (point value = 12)
king of clubs (point value = 13)
king of hearts (point value = 13)

isEmpty: false

Activity 3: Adding a Shuffle Method to the Deck Class

Complete the `Deck` class by implementing the `shuffle()` method.

Below is the algorithm you must use:

```
Given a deck of size n with indexes from 0 to n-1

For k = n-1 down to 1,
    Generate a random integer r between 0 and k,
    inclusive;
    Exchange cards[k] and cards[r].
```

Run `DeckTester` again. The deck should now be shuffled - perhaps like the output below, but it is random - so it will likely be different!

size = 6
jack of clubs (point value = 11)
king of clubs (point value = 13)
queen of clubs (point value = 12)
queen of hearts (point value = 12)
king of hearts (point value = 13)
jack of hearts (point value = 11)

isEmpty: false

Activity 4: Implementing the Elevens Board

Start by reading over the code in the abstract `Board` class. `Board` is abstract because it has one or more abstract methods. Recall that *interfaces* have all abstract methods. Neither interfaces of abstract class can be instantiated.

Now look at the `ElevensBoard` code. You need to complete 4 methods in this class.

Start by completing the `ElevensBoard` helper methods. These methods will be called only by other methods within the `ElevensBoard` class, which is why they are private.

- `boolean containsPairSum11(List<Integer> cardIndexes)`

  This method determines if the list `cardIndexes` contain a pair of cards whose point values add to 11.

  Important: The ArrayList `cardIndexes` is a list of **Integer** values representing positions of cards on the board. To get the `card` object at an index you must use the `cardAt(int k)` method in the `Board` class.

- `boolean containsJQK(List<Integer> cardIndexes)`

  This method determines if a list of `cardIndexes` contain a jack, a queen, and a king in any order.

- `public boolean isLegal(List<Integer> cardIndexes)`

  Determines if the `cardIndexes` form a valid group for removal. Either two cards that sum up to 11 -or- 3 cards consisting of a J, a Q, and a K.

  Use the "helper methods" containsPairSum11 and containsJQK to determine if the cards can be removed.

- `boolean anotherPlayIsPossible()`

  This method should also utilize the “helper methods”. It should be very short. Determine if there are any legal plays left on the board.

***** Test your code by running the main method in `ElevensGUIRunner` *****