CS230

LEARNING GOALS

- Using arrays to hold and maintain collection of objects
- Using loops to process array elements

EXERCISE: DECK OF CARDS!

Create a class named `Deck` that creates a Deck of Card objects.

Specifications

For this exercise you are **NOT allowed to use any of Java's Arrays class**. Just use arrays of Objects as we did in lectures.

0. Study and use the provided Card class. The Card.java (assign207/Card.java) class is provided for you. Read it very carefully and provide documentation for all the methods that ask you to "WRITE DOCUMENTATION". Pay close attention to the equals() method.

Create a `Deck` class that contains the following:

1. Instance variables

- a counter to keep track of how many cards are in the Deck.
- a constant to keep track of the total number of cards in a full Deck (52)
- an array to keep track of the collection of Cards in the Deck

2. Constructor

It should fill the array with all 52 unique possible Cards. Recall that a Card can have one of 4 possible suits and one of 13 possible values. You have complete freedom over how this can happen.

3. Instance methods

- Start with `toString()` : as usual, this method should provide a neat String representation of an object of type Deck. This representation should print all the Cards in the Deck.

- Instance method `cut()`. It picks at random a number between 1 and the number of cards in the Deck and it splits the deck in two piles. It then changes the order of these two piles in the Deck: the top pile goes to the bottom of the Deck. The Cards of the Deck remain the same but their order in the collection change to reflect the cut. For example, if the cards were A,B,C,D,E and we randomly picked 2, the Deck is rearranged so that the cards appear in this order: C,D,E,A,B.
- Instance method `shuffle()`. It picks at random a card from the Deck and adds it to a new pile repeatedly until there are no cards left in the Deck. Then, the new pile becomes the collection of cards in the Deck.
- The previous method will use two helper methods that you can name as you want. One method will find the index of the Card you are removing, and the second method will perform the removal (similar to how you removed an Animal from the Zoo).

4. No main() is required in the Deck class

Instead, create another client class called `Game.java`. This class contains only a `main()` method. It main purpose is to test your `Deck` class. Make sure you test your methods as you develop your code.

5. Write good javadoc Make sure you add nice javadoc to your code that includes:

- `above the class documentation` (but below any `import` statements): Provide a quick description of what this class is doing. Use the `@author` and `@version` tags.
- `above a method documentation`: For every method provide a short sentence to succinctly and accurately describe what this method is doing. Also, use the `@param` and `@return` javadoc tags as needed.
- `in-line comments` if needed: Add them only as appropriately, to document code that may be not easy to understand. You should strike for a balance here: Ideally, the reader of your code should be able to follow it by just reading the comments. At the same time, you should not document the obvious.
- Finally, take a look at the documentation automatically produced for your class in BlueJ, (click the Source Code button at the top right of the editing window and select Documentation) and examine it to make sure it is complete and reads well.

6. Testing

Save the printout of the `Game` class into a file named `GameTesting.txt`.

SUBMITTING YOUR CODE

- Submit your commented `Card.java`, your `Deck.java`, your `Game.java`, and your `GameTesting.txt`, to Gradescope.

Remember that in Gradescope you have to upload all the files at once. Every submission you do overwrites the previous one, so if you upload one file at a time, only the last will be saved.

Check your submission to make sure the right files have been submitted.

Happy Coding!