Information

Assignment 5: Arrays and Recursion (12%)

Instructions

Read all assignment directions and questions before you begin preparing your answers, and consult your Open Learning Faculty Member if you have any questions about the assignment.

This is not a timed assignment. It will not close until you **click "Finish attempt" and "Submit all and Finish"** button, at which point it will notify your Open Learning Faculty Member that your assignment is ready for marking.

You can leave and return from this quiz environment.

This assignment will be submitted as one .zip file in submission point question so you do not need to enter anything in text boxes below questions. Review the Assignment and Submission Instructions before submitting your assignment for grading.

Question **1**

Not yet answered

Not graded

As described in the textbook, an array is often used as a grouping structure for a set of related data items. Common operations on an array of numerical values are to find the minimum value, maximum value, and average value. To demonstrate these operations, design and implement a class called `ArrayOps`, which has an integer array. The constructor receives a parameter, which is the size of the array to be dynamically allocated in the constructor, then populates that array with random integers from the range -size through +size. Methods are required that return the minimum value, maximum value, average value, as well as a String representation of the array values. Document your design with a UML Class diagram that is to be submitted along with the testing exhibits. Create a separate driver class that prompts the user for the sample size, then instantiates a `ArrayOps` object of that size and outputs its contents and the minimum, maximum, and average values.

Hint: Obviously the minimum and maximum values will be integers, but the average will need to allow decimals.

Testing: Include the output for at least five (5) different test runs of various sizes, that shows the array contents along with its minimum, maximum, and average values. Make sure you manually check the results, especially the average.

(Ignore the text box below, your submission can be uploaded to the final question)

Question **2**

Not yet answered

Not graded

A standard playing card can be modelled with the following class, where the face values are 1 thru 13 (being Ace thru King), and the suit values are 1 thru 4 (being Diamonds, Clubs, Hearts, Spades):

```
public class Card {

  private int face;

  private int suit;

  public Card (int f, int s) {

    face = f;

    suit = s;

  }

  public String toString() {

    return face + “ of ” + suit;

  }

}
```

Complete this design so that the toString() method returns a nice textual representation, e.g. Ace of Diamonds instead of 1 of 1. This logic should use arrays to provide a lookup of the appropriate text name.

Use this completed class to design and implement a class called `CardDeck` that simulates a standard deck of playing cards. The design should store 52 objects of the Card class using a one-dimensional array. The constructor should populate the deck. Include methods to shuffle the deck, deal a card, return the number of cards left in the deck, and a `toString` to show the contents of the deck. Document your design with a UML Class diagram, that is to be submitted along with the testing exhibits. Create a separate driver class that first outputs the populated deck to prove it is complete, shuffles the deck, and then deals each card from a shuffled deck, displaying each card as it is dealt along with the number of cards left in the deck.

Hint: The constructor for `CardDeck` should have two nested for loops for the face values (1 to 13) within the suit values (1 to 4) calling the Card constructor. The shuffle method does not have to simulate how a deck is physically shuffled; you can achieve the same effect by repeatedly swapping pairs of elements in the array with the indexes chosen at random.

Testing: Include two complete runs to demonstrate the random effect of shuffling.

(Ignore the text box below, your submission can be uploaded to the final question)

Question **3**

Not yet answered

Not graded

Design and implement a recursive program to determine and print up to the Nth line of Pascal's Triangle, as shown below. Each interior value is the sum of the two values above it.

```
                                1
                            1       1
                        1       2       1
                    1       3       3       1
                1       4       6       4       1
            1       5      10      10       5       1
        1       6      15      20      15       6       1
    1       7      21      35      35      21       7       1
1       8      28      56      70      56      28       8       1
```

Hints: You might use an array to hold the values for a given line, although that is not necessary. It is not necessary to format the output exactly as presented above.

One recursive approach is:

T(r, 0) = T(r, r) = 1

T(r, c) = T(r - 1, c - 1) + T(r - 1, c)

Include at least three test runs of different sizes.

(Ignore the text box below, your submission can be uploaded to the final question)