AP ACTIVITY **4.2.2**

Design a Deck Class (AP)

INTRODUCTION

A deck of cards can be used for many different purposes, such as building a house of cards, doing magic tricks, and of course, playing card games. When you think about designing a class to represent something in the real world, like a deck of cards, you must also have in mind the general purpose of the class. But if you get too general of a purpose, the design can become too broad and have little meaning for your project.

Materials

- Computer with BlueJ IDE

Procedure

Part I: Design the Deck Class

1. Consider implementing a class to represent an entire deck of cards for playing a card game like Sevens or Elevens. Describe its instance variables and methods, and discuss your design with a classmate.
2. Using the source code your teacher provides, open and create an *ElevensActivity2* project in BlueJ.

> **Reminder**: To create a BlueJ project as you did in AP Activity 4.2.1, open and create an *ElevensActivity2* project in BlueJ for the Java files that have been provided to you:
>
> - In BlueJ, select **Project** > **Open Non BlueJ ...**
> - In the navigation window, navigate to your *ElevensActivity2* directory. The folder will look empty.
> - Click **Open** in BlueJ. This creates a BlueJ project in a new package.bluej file.

3. Read the partial implementation of the `Deck` class. This file contains the instance variables, constructor header, and method headers for a `Deck` class general enough to be useful for a variety of card games.

4. Discuss the `Deck` class with your classmates; in particular, make sure you understand the role of each of the parameters to the `Deck` constructor, and of each of the private instance variables in the `Deck` class.

5. Describe your `Deck` design elements, specifically, the constructor's parameters and all instance variables needed in the class.

Part II: Create the Deck Class

6. Complete the implementation of the `Deck` class:

   a. `Deck` constructor — This constructor should receive three arrays as parameters. The arrays will contain the ranks, suits, and point values for each card in the deck. The constructor must create an ArrayList, and then create the specified cards and adds them to the list. For example, if `ranks = {"A", "B", "C"}`, `suits = {"Giraffes", "Lions"}`, and `values = {2,1,5}`, the constructor would create the following cards:

```
["A", "Giraffes", 2], // first card
["B", "Giraffes", 1], // second card
["C", "Giraffes", 6], // ...
["A", "Lions", 2],
["B", "Lions", 1],
["C", "Lions", 5]    // sixth card
```

   The parameter size of the ArrayList would then be set to number of cards created, which in this example is 6. Finally, the constructor should shuffle the deck by calling the `shuffle` method. Note that you will not be implementing the shuffle method until AP Activity 4.2.4.

   b. `isEmpty` — This method should return `true` when the size of the deck is `0`; `false` otherwise.

   c. `size` — This method returns the number of cards in the deck that are left to be dealt.

   d. `deal` — This method "deals" a card by removing a card from the deck and returning it, if there are any cards in the deck left to be dealt. It returns `null` if the deck is empty. There are several ways of accomplishing this task. Here are two possible algorithms:

Algorithm 1: Because the cards are being held in an `ArrayList`, it would be easy to simply call the `List` method that removes an object at a specified index, and return that object. Removing the object from the end of the list would be more efficient than removing it from the beginning of the list. Note that the use of this algorithm also requires a separate "discard" list to keep track of the dealt cards. This is necessary so that the dealt cards can be reshuffled and dealt again.

Algorithm 2: It would be more efficient to leave the cards in the list. Instead of removing the card, simply decrement the size instance variable and then return the card at size. In this algorithm, the `size` instance variable does double duty; it determines which card to "deal" and it also represents how many cards in the deck are left to be dealt. This is similar to the algorithm you used in Lesson 4.1 and **it is the algorithm that you should implement**.

7 Once you have completed the Deck class, modify `DeckTester.java`. Add code in the `main` method to create three `Deck` objects and test each method for each Deck object.

CONCLUSION

1. Explain in your own words the relationship between a deck and a card.
2. Consider the `deck` initialized with the statements below. How many cards does the deck contain?

```
String[] ranks = {"jack", "queen", "king"};
String[] suits = {"blue", "red"};
int[] pointValues = {11, 12, 13};
Deck d = new Deck(ranks, suits, pointValues);
```

3. The game of Twenty-One is played with a deck of 52 cards. Ranks run from ace (highest) down to 2 (lowest). Suits are spades, hearts, diamonds, and clubs as in many other games. A face card has point value 10; an ace has point value 11; point values for 2, ..., 10 are 2, ..., 10, respectively. Specify the contents of the `ranks` , `suits` , and `pointValues` arrays so that the statement

   ```
   Deck d = new Deck(ranks, suits, pointValues);
   ```

   initializes a deck for a Twenty-One game.
4. Does the order of elements of the `ranks` , `suits` , and `pointValues` arrays matter?