Programming on Purpose (CSC 120)
Winter 2024

Lab 7 – Hit the Deck!

2/15/2024

Objectives

- More practice with Java.
- More practice with information hiding. You can have vastly different implementations of a class (how methods work) and still have the same behavior (what methods do)
- See how your choice of implementing one method can affect the implementation of another

Preliminary Setup

For this lab, you will be creating a class that models a deck of cards. And you will think about two different ways of implementing this model, while the behavior of these two implementations will be the same.

1. Download the starter code.
2. You will write all of your code in `Deck.java`. So go ahead and add an `@author` tag with your name.
3. Read the `Card` class, which is already finished. Notice the *overloaded* constructor which lets you make `Card` objects in different ways.
4. Have a quick look at `DeckSandbox.java`. It contains some code to help you test and debug your implementation.

1 Step 1: Complete the Deck

This class models a deck of cards. It should support dealing a card, shuffling the deck, and gathering all cards up again after some have been dealt. Start by reading the javadocs for the methods corresponding to those actions so that you understand what their intended behavior is.

The Deck class already has the instance variables you need. You are not allowed to change them or make any more of them. **Before you start programming, think about and make a plan for how you are going to use these instance variables to achieve the desired behavior.** Write down your plan in a text file `reflection.txt` that you add to your lab folder. You can do this in VS Code.

Then, complete the following methods:

1. Finish writing the constructor.
2. Write the `size` method.

3. Write the `isEmpty` method. Reusably. (I.e. this method should not directly access instance variables.)
4. Write the `deal` method. This method either returns null if the deck is empty, or removes and returns the top most card.
5. Write the `shuffle` method. This method should iterate through the indices of the (remaining) cards in the deck. For each index, swap the card at that index with a card at a random index.
6. Write the `gather` method which returns the deck to a state where all cards are undealt. It does **not** have to return the deck to an unshuffled state.

2 Step 2: Reflect

Answer the following questions by adding to your `reflection.txt` file.

1. What is the worst-case runtime in big-O notation for the following methods: `deal`, `shuffle`, verb—gather—?
2. Now imagine that the starter code had asked you to *only* use the following two instance variables (besides `rand` and the constants):

   ```
   // the cards in the deck
   private Card[] theCards;
   // whether or not the deck is ordered (false) or randomized (true)
   private boolean shuffled;
   ```

   Describe in pseudocode how the following methods could be implemented using only those two instance variables:

   - `deal`: This method either returns null if the deck is empty or removes and returns a card. If the deck is in order, it should return the next card in order. If the deck is shuffled, it should return a random card. (Note that it is this method that does the "shuffling" work by picking a card at random when it's time to deal a card from a shuffled deck!)
   - `shuffle`: Based on what `deal` is doing, what does this method have to do?
   - `gather`: Returns the deck to contain all 52 cards. (Note that this method should not create a new `Deck` *object*. It should restore the already existing `Deck` object.)
3. What is the worst-case runtime in big-O notation for your new implementations of `deal`, `shuffle`, and `gather`?

3 Revise your code

Before turning in any program in this class, remember this mantra:

> Just because it works doesn't mean it's good.

Part of your grade will also come from things like how understandable and readable your code is. You will also be graded on the neatness, presentation, and style of your program code.

Make sure that all modules and functions are documented (even those that you didn't write).

Don't forget to cite who you received help from and include the honor code affirmation in the javadocs of each class that you wrote or modified:

```
I affirm that I have carried out my academic endeavors
with full academic honesty. [Your Name]
```

4 How to submit

Submit the project by uploading `Deck.java` and `reflection.txt` to Gradescope.

Make sure to add your partner to the submission so that you both have access to it through Gradescope.