INTRODUCTION TO COMPUTER SCIENCE II
PROJECT 2
Testing shuffles

We are going to use the concepts *abstract classes* and *inheritance* so that we may experiment with *shuffling algorithms* and *pseudo-random number generators*.

1 `Decks` that shuffle differently

Consider the following (non-comprehensive) approaches to shuffling a deck of `Cards`:

- **Ideal:** Using one of the techniques described in class that, we believe, provides a uniform probability for each card landing in each position.
- **Null:** Don't shuffle at all. This is, of course, a pathalogically poor way to generate a random permutation. Useful as a point of comparison.
- **Brute force:** For some number of repetitions, randomly pick two `Cards` from the deck and swap their positions. The quality of the shuffle will depend strongly on the number of repetitions.
- **Human:** An idealized simulation of how humans shuffle. That is, divide the cards in the deck in half by calculating the halfway point (e.g., for a full deck, all cards from positions 0 to 25 are in the first half, while those in positions 26-51 are in the second). Then, "zip" the cards together by alternating cards from each half. Repeat some number of times, where (again) the number of repetitions will dictate the quality.

Any code that uses a `Deck` object should not need to know which type of `Deck` (i.e., what kind of shuffling that `Deck` performs). So, if the `Deck` class were *abstract*, as was its `shuffle()` method, then:

- each type of shuffle could be implemented in a *concrete subclass* of `Deck` that implemented the `shuffle()` method in the appropriate manner, and
- the code that used a `Deck` could simply take a `Deck` pointer to whichever concrete subclass it wants, performing the same operations irrespective of the deck's type.

2 Measuring the shuffle

If we have different methods for shuffling, we want to test them to see how well (or poorly) they work. So, we can write a class, `ShuffleTest`, which contains a `main()` method that begins the process of testing the different types of `Decks` and their shuffling methods. Specifically, this class would:

- Create a concrete object from one of the `Deck` subclasses, keeping a `Deck` pointer to it.

- Randomly select a position in the deck (still in its original order), and keep track of which `Card` occupies that position.
- Shuffle the deck.
- Draw cards from the deck until the randomly selected card is found, thus determining the final position of the card.
- Calculate the *distance* between where that card started (before the shuffle) and where it ended (after). The distance should be the number of positions that the Card moved **forward** in the array, assuming that the position numbers conceptually "wrap around" to position zero.

The `ShuffleTest` program should keep a *histogram* of distances. That is, given that each card may move between 0 and 51 positions when shuffled, the program should keep an array of 52 integers such that the entry at position k represents the number of times the measured distance was k.

2.1 Invoking the program

`ShuffleTest` should be run like so:

```
  $ java ShuffleTest 100 Brute 5000
```

Specifically, the three parameters are:

- **Number of data points:** The number of times the above process of finding a *distance* should be performed and recorded. This parameter is independent of the choice of shuffle type.
- **Shuffle type:** Which kind of shuffle to use and test.
- **Repetitions per shuffle:** For those shuffling types that require this parameter, the number of repetitions to perform. For example, the *Human* form of shuffling needs to be repeated some number of times to constitute a full shuffle. This value should indicate how many such repetitions are performed **per call to the `shuffle()` method**.

As a result, this program should emit its histogram, like so:

```
0 103
1 95
2 112
3 87
...
51 100
```

That is, the first column shows a *distance*, while the second column indicates the *number of instances of that distance observed*.

3 Your assignment

Write the above. Then, for each type of shuffle—and for parameterized shuffles, for a variety of repetitions—record the distance distributions observed. For each, then perform a χ^2 test to determine how well the shuffle (with a given number of repetitions, if applicable) yielded a uniform distribution.

4 How to submit your work

Use the CS submission systems to submit your work. Specifically, you will need to submit all of your source code files. You may use either of the following two methods, while connected to `remus` or `romulus`, to use the submission system:

- **Web-based:** Visit the submission system web page.
- **Command-line based:** Use the `~lamcgeoch/submit` command at your shell prompt.

This assignment is due on Tuesday, Nov-15, 11:59 pm.