Harrison Amyotte

String Randomization

User Story - Function that accepts a string and then returns a string containing the same characters as the input string – but in a random order.

Implementations

A note on data

An additional class with 2 public string arrays of 500 words is included. The first array is a random sampling of english words. The second is a set of random strings, 20 characters, upper and lower case letters, as well as numerals.

(Each implementation used the same global Random class, to allow for comparison on results)

1. **1-1 Swap**

This method runs through a loop of the length of the input string. In this loop it will randomly generate an integer between 0 and length of array. The index of the string at the random number will be swapped with the n-th value of the array.

2. **LINQ**

This method will convert the input to a new char array, while using an extension method (OrderBy) to order the array to a random number.

3. **Recursion**

This method accepts an int (length of string), and recurs until this counter has hit 0. The method makes use of 2 global strings. One is set to the input, the other is set to the resulting character returned from recursion, which is then returned in the main method.

Method Comparisons

First each method was printed out to compare the word and the randomization of the word. This was used to compare performance according to the user story.Then, to test for efficiency, each method was run during a .NET stopwatch.

	Speed (*milliseconds*)	**Randomness**
Shuffle	0-1 O(n)	Random
LINQ	25-30 O(n logN)	Pseudo-random
Recursion	1-3 O(n)	True Random

The results paint an interesting, and surprising picture. I was not expecting recursion to be as close as it was to the shuffle method in speed, due to the use of the global public variables. I also was not expecting the results from the LINQ method to have such low-random results.

Harrison Amyotte

Use Cases

Questions

If this was a real world situation, i would ask the following questions to get more context behind what i’m working on. For example, if this was scrambling production data for security reasons, the way I am seeding my randomization would need to be changed.

A. Is it necessary to provide a unique randomization of the string for each operation?
B. Is it necessary to 'fully' randomize the string - or can approximation be used?
C. What is the frequency of requests to this method?
D. Does any concern need to be given to data sanitization on the inputs?
E. Is the randomization due to a security effort?

The method chosen would differ based on the surrounding functionality, as well as the questions listed above. That being said, I would use the shuffle method for most cases. It was the fastest, and produced acceptable random results. It is very insular, and is decently readable.

However the argument could be made to use recursion. If there were additional calculations or calls needed per loop, recursion could break these down very simply. In this case, the shuffle method would slow down considerably, unless it was applied under asynchronous operation. Asynchronous isn’t always possible if a database needs to be updated with the results, as out of order exceptions would occur, if not context issues, depending on the amount of people calling this method and the technology used to connect to the database.

There is not much to be said for the LINQ method, as the speed was so much slower than the others, as well as the less than stellar randomness (which could be improved at the loss of more speed). If the volume of calls is sufficiently low, LINQ does offer a number of benefits. First, it's a single line, and super easy to read and understand. The code could be reused or integrated into larger logic bodies very easily. Additionally, if the data is sourced from a database, LINQ could be an extension method on that call, reducing run time. Other methods would need an intermediary to pull that data, increasing time.