Julian Moon  
June 6, 2018

String Encryption/Decryption

Abstract:

The purpose of this project is to explore ways to convert various objects into different data types, as well as practice developing some basic mathematical algorithms to transform them. The program takes in a String to encrypt and provides multiple ways to transform (encrypt) it, during which it also converts it into different data types. It then returns a custom class of object called “Encrypted,” which contains the encrypted version of the String, as well as a decryption key. The decryption component of the code takes in the Encrypted object and returns the original String by reading the decryption key and performing appropriate inverse operations.

Structure:

The program has three classes: Encryption, Decryption, and Encrypted. Both Encryption and Decryption don’t serve as objects, but merely as containers for static methods associated with each process. The Encrypted class serves as an object class that has the encrypted message (List of integers) and the decryption key (String) as fields. The class only contains straightforward accessor and mutator methods used for modifying the fields.

Method:

The Encryption process is in four steps, namely, translation, random mapping, base change, and permutation. The String to be encrypted is broken down into a List of characters, and can be translated, meaning every character is shifted back some number within the alphabet (non-alphabetic characters are exempted). Random mapping maps each of the characters to an integer according to a randomly generated map. The map is bijective (meaning every element is mapped and mapped to a unique element) which is necessary for an inverse mapping to be created. Note that the random mapping step always happens, whereas the three other steps are voluntary. Then the List of integers can be transformed into another base (< 10), and finally, the List of integers can be permuted using a randomly generated permutation sequence. The permutation idea originates from permutation groups in mathematics, and just like elements in permutation groups having an inverse element, the decryption algorithm generates the inverse element of the encryption permutation so that the List can recover its original order. While encrypting, the code also generates a decryption key as a String, which along with the encrypted List of integers, is stored in an Encrypted object. The decryption method in Decryption takes this object, and by reading the key with a scanner, determines the appropriate set of decryption methods it needs to use to return the original String. The Decryption methods are straightforward and do not require much explanation, as they are simply inverse operations of the Encryption class. One addendum in the Decryption class however, is the addition of a set of methods that would convert Strings from the key to objects used for decryption.

Accomplishments:

The project contains multiple methods that employ recursion to perform certain mathematical operations. I was surprised that I was able to immediately solve these problems and do so using recursion, as I had quite some trouble not too long ago while solving similar problems. In addition, I successfully integrated a scanner in the project and handled any possible bugs that may result from inappropriate inputs (such as entering a character when an integer needs to be passed).

Potential Improvements:

First, a word about efficiency. Everything works in O(N) time as most of the conversion or transformation in the encryption/decryption process simply goes through the characters or integers while transforming them. The permutation step however, is highly inefficient as it requires the use of set() method (which works on O(N)) in the List interface N times making the method work on , which results in the permutation method becoming relatively slow for large N’s. The usage of LinkedList rather than ArrayList would not help in terms of efficiency for this method as it would run into the same issue (I read that ArrayList is also more efficient for random access anywhere within the list, making it a better choice over LinkedList in this case).

Secondly, the code has limitations in the types of Strings it can encode. At first, the project targeted only characters in the alphabet. This seemed rather impractical, so I added common punctuation marks as well as numbers. However, this still does not cover everything, and as such, the program could be further generalized. However, considering that the code encrypts a message, I would guess that its use would be limited to relatively concise and simple messages that don’t contain odd characters.

Finally, the concept of encryption sounded rather elaborate when I came up with the idea, but the entire encryption process turned out to be a bit too simple. I looked up actual data encryption methods and found that encryption is actually done bit-by-bit. I have never studied cryptography, but have heard about its relationship to mathematics while I studied group theory. The rather simplistic nature of the encryption makes this project not viable in practical use, but nevertheless provided a great opportunity to refine many different aspects of my coding skills.