Using Abstract Data Types – 70 course points

This assignment consists of two parts to exercise your ability to work with Strings and recursion.

Programming

Write 2 programs and submit on Sakai.

We provide a zip containing *RecusiveAppend.java*, and *RunLengthEncoding.java*. For each problem update and submit the corresponding file.

DO NOT use System.exit().
DO NOT add the project or package statements.
DO NOT change the class name.
DO NOT change the headers of ANY of the given methods.
DO NOT add any new class fields.
ONLY display the result as specified by the example for each problem.
DO NOT print other messages, follow the examples for each problem.
You may USE the StdOut library.

1. *Recursive transform (35 points)*. On *RecursiveAppend.java* write a recursive method appendNTimes that receives two arguments, a string and an integer. The method appendNTimes returns the original string appended to the original string n times.

   Use the following method header:
   public static String appendNTimes (String original, int n)
   Examples:
   appendNTimes("cat", 0) returns “cat”
   appendNTimes("cat", 1) returns “catcat”
   appendNTimes("cat", 2) returns “catcatcat”
   The following restrictions apply to method appendNTimes:

   1. **YOUR CODE MUST BE RECURSIVE**
   2. Do not use loops (while, do/while, or for).
   3. Your code must return a string without extra space, comma or any other character that is not in the original string

   You may write your own main method to test your appendNTimes method. The Autograder will ignore your main method.

2. *Run-length encoding (35 points).* Data compression is used behind the scenes in computer systems quite often, computer files and other kinds of data can be compressed to a smaller size for easy storage or transportation. Later, they are decompressed and used in their original form. One basic idea is to find parts of

the data that are identical to each other and use some kind of trick to describe that more efficiently.

Run-length encoding (RLE) encodes a run of repetitions with the length of that run. RLE is a simple compression algorithm (an algorithm which takes a block of data and reduces its size, producing a block that contains the same information in less space). It works by replacing repetitive sequences of identical data items with short tokens that represent entire sequences. Applying RLE to a string involves finding sequences in the string where the same character repeats. Replace each such sequence by a token consisting of:

1. The number of characters in the sequence
2. The repeating character

If the character does not repeat, it appears as a single character in the compressed string with no number preceding it.
For example, consider the following string:

qwwwwwwwwweeeeerrtyyyyyqqqqwEErTTT

After applying the RLE algorithm, this string is converted into:

q9w5e2rt5y4qw2Er3T

In the compressed string, “9w” represents a sequence of 9 consecutive lowercase “w” characters. “5e” represents 5 consecutive lowercase “e” characters, etc.
Write a RLE library by implementing the follow API:

```
public class RunLengthEncoding {

  // Encodes the original string by finding sequences in the string
  // where the same character repeats.
  // Replace each such sequence by a token consisting of: the number
  // of characters in the sequence followed by the repeating character.
  // Write an iterative encode method.
  // Returns the encoded string.
  public static String encode (String original)

  // Decodes the original string encoded with the encode method.
  // Returns the decoded string.
  // YOUR decode METHOD MUST BE RECURSIVE.
  // Do not use while, do/while, or for loops.
  public static String decode (String original)

  // Tests each of the API methods by directly calling them.
  public static void main (String[] args)
}
```

For decode, you may assume that the character counts will be single-digit numbers (a character will not repeat more than 9 times consecutively).
Hint #1: remember that characters are represented by numeric codes. You can decrement a character variable as follows:

```
char c = '7';
c--; // c will now hold the character '6'
```

Hint #2: You can check if a character is a digit by using the isDigit() method from the Character class as follows:

```
char c = '7';
Character.isDigit(c); // returns true
```

Hint #3: You probably will not need to use this hint for this problem. However, a fast way to convert a digit character into the numeric value of the digit is to subtract the character code for the digit zero:

```
char c = '7';    // this has the character code 55, not 7
int x = c - '0'; // this produces the number 7
```

Before submission

1. *Collaboration policy*. Read our collaboration policy here.
2. *Update @author.* Update the @author tag of the files with your name, email and netid.
3. *Submitting the assignment*. Submit *RecursiveAppend.java,* and *RunLengthEncoding.java* separately on Sakai.