1. Write and test a method `reverseString` that takes a `String` as its argument and constructs
and returns a `String` containing the same `char` values, but in the opposite order. (For efficiency,
use a `StringBuilder` object to collect the `char` values as you extract them from the given string,
then convert the `StringBuilder` to a `String` at the end.) Test the method with "Hello, world!".  
The method should return "!dlrow ,olleH".

The `StringBuilder` class has the built-in `reverse` method. Do not use this method for
answering the exercise!

2. (3 points) One traditional way of lightly disguising the content of a text written using the
twenty-six-letter English alphabet is “rot13 encryption,” in which every letter is replaced by the
letter thirteen positions before or after it in the alphabet. In this cryptosystem, ’A’ is replaced by
’N’, ’B’ is replaced by ’O’, and so on; after the midpoint of the alphabet, you back up instead
of counting forwards, so that ’N’ is replaced by ’A’, ’O’ by ’B’, and so on through ’Z’, which
is replaced by ’M’. In social media, rot13 encryption is occasionally used to conceal (briefly) the
answers to riddles and trivia questions.

Write and test a method called `rot13` that takes a `String` as argument, applies rot13 en-
cryption to each of the letters of the English alphabet that occurs in that `String` (leaving other
characters unchanged), and returns the result. Capital letters should be encrypted as capital letters
and lower-case letters as lower-case letters. For example, the method call `rot13("Hello, world!")`
should return "Uryyb, jbeyq!"

3. Explain why there is no need to implement a separate method for decrypting strings that
have been encrypted with rot13.

4. The `format` method in the `String` class generates a `String` result from a “format string”
given as the first argument to the method) by embedding string representations of the subsequent
arguments, which can be of various types. As in a call to the `printf` library function in C, the
format string contains “format specifiers” that identify the data types of the values to be printed,
control field width, precision, justification, and so on. (Java being what it is, the format specifiers
can have even more bells and whistles than in C, so study them carefully and try some experiments
if your reading leaves any uncertainties in your mind.)

Write a Java program that prints out a multiplication table from 1 to 10. The program needs
to display ten numbers per line, where each number is right-justified into a field of width 5.

5. (1 point) The `substring` method in the `String` class returns a new string that is a
substring of a given string. This method takes two arguments. These arguments are the start
index and end index of characters within the original string. Note that the substring does not
include the character at the end index.

Write and test a method called `countSubstring` that counts and displays the number of
substrings that match a target string using the `substring` method. Consider a string
"this and this and that and this"

and a target string "this". In this example, there are 3 substrings. Do not use `startsWith`
method for this exercise.
This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit

http://creativecommons.org/licenses/by-sa/4.0/deed.en_US