CSCI212_12 Spring 2019
Assignment 1

This assignment can be submitted as many times as you want. However, only the last submission will be graded. The earlier versions will not be graded. So make sure that your final submission works and produces the correct results. Please do not cheat or copy.

1. The standard quadratic equation That you learned in high school

   $ax^2 + bx + c = 0$

   has two solutions given by the formula

   $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

   (a) The first solution is obtained by using + in place of ±.

   (b) The second is obtained by using – in place of ±.

   Most of this expression contains simple operators covered in Chapter 3 of your textbook. The one piece that's missing is taking square roots, which you can do by calling the standard function Math.sqrt. For example, the statement

   ```
   double y;
   y = Math.sqrt(x);
   ```

   sets y to the square root of x.

   Write a Java Program that accepts values for a, b, and c, and then calculates the two solutions (which may both be the same). If the quantity under the square root sign is negative, the equation has no real solutions, and your program should display a message to that effect. You may assume that the value for a is nonzero. Your program should be able to duplicate the following sample run:

   Enter coefficients for the quadratic equation:
   a : 1
   b : 5
   c : -6

The first solution is -6.0
The second solution is 1.0

2. Here's a little game to play. Starting with any positive integer n form a sequence in the following way:

   - If n is even, divide it by 2 to give $n' = n/2$.
   - If n is odd, multiply it by 3 and add 1 to give $n' = 3n + 1$.

   Then take n' as the new starting number and repeat the process. For example, $n = 5$ gives the sequence

   $$5, 16, 8, 4, 2, 1, 4, 2, 1, ...$$

   and $n = 11$ gives the sequence

   $$11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, ...$$

   Sequences formed in this way are sometimes called hailstone sequences because they go up and down just like a hailstone in a cloud before crashing to Earth. However, it seems that all hailstone sequences eventually end in the endless cycle

   $$4, 2, 1, 4, 2, 1.$$

   The ones for $n = 5$ and $n = 11$ both do, though other values for may n generate a very long sequence before the repeating cycle begins.

   Write a Java Program that reads in a number from the user and then displays the Hailstone sequence for that number, followed by a line showing the number of steps taken to reach 1. For example, your program should be able to produce a sample run that looks like this:

```
This problem computes the Hailstone sequences.
Enter a positive integer: 15
15 is odd, so I make 3n+1 = 46
46 is even, so take half = 23
```

```
23 is odd, so I make 3n+1 = 70
70 is even, so I take half = 35
35 is odd, so I make 3n+1 = 106
106 is even, so I take half = 53
53 is odd, so I make 3n+1 = 160
160 is even, is I take half = 80
80 is even, so I take half = 40
40 is even, so I take half = 20
20 is even, so I take half = 10
10 is even, so I take half = 5
5 is odd, so I make 3n+1 = 16
16 is even, so I take half = 8
8 is even, so I take half = 4
4 is even, so I take half = 2
2 is even, so I take half = 1
The process took 17 steps to reach 1.
```

The fascinating thing about this problem is that no one has yet been able to prove that it always stops. The number of steps in the process can certainly get very large.

3. Suppose you have managed to intercept a message that reads as follows:

```
:mmZ\dxZmx]Zpgy
```

The message is encrypted using a secret code. You have just learned that the encryption method is based upon the ASCII code (Appendix 7 in your textbook). Individual characters in a string are encoded using this system. For example, the character 'a' is encoded using the number 97 and 'b' is encoded using the number 98. The code depends on a Secret key which is an integer value. This secret code takes each letter of the message and encrypts as follows:

```
if(originalChar + key > 126) then
   EncryptedChar = 32 + ((originalChar + key) - 127)
else
   EncryptedChar = (originalChar + key)
```

For example, if one uses Key = 10 then the message "Hey" would initially be represented as

Character	ASCII code
H	72
e	101
y	121

And "Hey" would be encrypted as:

```
Encrypted H = (72 + 10) = 82 = R in ASCII
Encrypted e = (101 + 10) = 111 = o in ASCII
Encrypted y = 32 + ((121 + 10) - 127) = 36 = $ in ASCII
```

Consequently, "Hey" would be transmitted as "`Ro$`".

Write a Java program that decrypts the intercepted message. You are only told that the key used is a number between 1 and 100. You can assume that the original message consists entirely of ASCII codes that represent only printable characters. Your program should try to decode the message using all possible keys between 1 and 100. When you try the valid key, the message will make sense. For all other keys, the message will appear as gibberish. Since there are only 100 keys this would obviously be a pretty crummy encryption system. Use the String method charAt() to handle each character in the String. This problem requires you to explore a bit on your own how to convert from a char to a number, process the number and then convert it back to a char. You cannot use the encryption method for decryption. You should first find the decription formula for the two cases you are given, and then test using your formula on the paper to see if it works on a single word that you first encript and using your formula decrypt and see if you can get back your original word.

There is a \ in the encrypted code. You need to escape encode it by using \\ if you hard code it in your program. This method should return a String.

Create three different methods. Each method should perform one of the above tasks. Then create a main method which will directly call the above

mentioned three methods. Print the results in the main method. You obtain user input in the main method. Each of your methods should just do the actual tasks and return the results. Submit all four methods in one class.