Programs for this lab should be stored in your Eclipse repository. Create a new Java project in your workspace called lab 9 . Remember to use a location on your P: drive or a USB drive (not the C: drive). When your programs are running correctly, turn in a printout of the Java code and send an email containing your .java files for each program in lab 9 (as *separate* file attachments) to the instructor at stlauterburg@salisbury.edu with the subject line "COSC 117 Lab 9".

Problem 1

Write a program called CaesarLite that encodes a message using a Caesar cipher (https://en.wikipedia.org/wiki/Caesar_cipher). The user should be prompted for the number of positions to shift (positive or negative) and a message to encode. For example, a shift of 3 would map A to D and B to E. You can assume that only capital letters need to be shifted and that other characters will not be changed. For this version, you can assume that the shift never goes past the beginning or end of the alphabet.

Example 1:

```
Enter shift: 4
Enter message: HELLO
Encoded message: LIPPS
```

Example 2:

```
Enter shift: -1
Enter message: IBM 9000
Encoded message: HAL 9000
```

Problem 2

Write a program called PowersOfN that will read an integer *n* from the user and create a two-dimensional array with ***n*** rows and 4 columns. The program should generate the numbers 1 through *n* in the first column, the squares of those numbers in the second column, cubes in the third and fourth powers in the fourth column. Do **not** hardcode the values for the array in your program. You should calculate the powers of ***n*** to be displayed as you need them. Display the results to the console as a table, *aligning the results in nice even columns* using the printf method. For example, if the user typed in a 5 the array would look something like the following:

```
 N      2      3      4
======================
 1      1      1      1
 2      4      8     16
 3      9     27     81
 4     16     64    256
 5     25    125    625
```

Problem 3

Write a program that simulates multiple rolls of **two** 6-sided dice (use **two** random numbers for this) and computes the percentage of the total number for each possible outcome. The program should ask the user how many times to roll the dice and use a loop to simulate that many rolls. The program *must* use an `int` array to keep track of how many times each possible roll (i.e., 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12) occurs. You may also find it convenient to use `double` arrays to handle the observed and expected percentages. After all the "rolls" have been performed, determine what the actual *observed* percentage of the total number of rolls was for each number. For example, if the dice were rolled 500 times and the value 8 was observed 62 times, then the *observed* percentage would be 12.40%. However, the *expected* percentage for 8 is 13.89% (5 out of 36 rolls) – you can find the percentages for other rolls online at several websites, e.g., (http://boardgames.about.com/od/dicegames/a/probabilities.htm). For each possible roll, display the roll value, the observed percentage, and the expected percentage. Display the results to the console as a table, *aligning the results in nice even columns* using the `printf` method. For example:

```
Enter the number of rolls to be simulated: 1300

The percentages observed for 1300 rolls:
roll    observed    expected
----    --------    --------
  2       2.83%       2.78%
  3       6.00%       5.56%
  4       8.12%       8.33%
 ..        ...         ...
 12       2.51%       2.78%
```

The program on this page illustrates the use of two-dimensional arrays:

```
public class TwoDimArrays {

    public static void main(String[] args) {

        // creating 2D arrays (3 rows by 6 columns)
        int[][] aa = new int[3][6]; // initializes cells to 0
        int[][] bb = { { 1, 2, 3, 4, 5, 6 },
                       { 7, 8, 9, 10, 11, 12 },
                       { 13, 14, 15, 16, 17, 18 } };

        System.out.println(tableSum(aa));
        System.out.println(tableSum(bb));
    }

    private static int tableSum(int[][] table) {
        int sum = 0;
        for (int row = 0; row < table.length; row++) {
```

```
                for (int col = 0; col < table[row].length; col++) {
                    sum = sum + table[row][col];
                }
            }
            return sum;
        }
    }
```