Programs for this lab should be stored in your Eclipse repository. Create a new Java project in your workspace called labA. 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 labA (as *separate* file attachments) to the instructor at stlauterburg@salisbury.edu with the subject line "COSC 117 Lab A".

Why lab A? The hexadecimal (base-16) number A is equivalent to the decimal number 10.

Problem 1

Write a program called Caesar that encodes messages using an *improved* Caesar cipher. As in the previous problem, the user should be prompted for the number of positions to shift and a message to encode. For this version, however, shifting beyond the end (or beginning) of the alphabet wraps around to the beginning (or end). For example, a shift of 3 would map **A to D**, **B to E**, and **Y to B**. And a shift of -3 would map **A to X**, **B to Y**, and **Y to V**. You can assume that only capital letters need to be shifted and that other characters will not be changed.

Example 1:

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

Example 2:

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

Problem 2

Create a program called **FootballScore** that allows the user to record the scores for each team for each quarter of a football game. The program should include a **one-dimensional array** that will hold the team names, and a **two-dimensional array** with *two rows and four columns* that will contain the number of points scored by each team in each of the 4 quarters. Use the arrays in parallel to match up team names with team scores, i.e., the first team in the team array will have its quarterly points in the first row of the points array. When the program is run, the user will input the team names and their scores for each quarter of the game. The program will then call a separate method called outputResults that accepts the two arrays as parameters and outputs the final game scores in appropriately aligned columns (see sample run below).

Sample run:

```
Enter the name of the 1st team: Ravens
Enter the name of the 2nd team: Steelers
Enter the 1st quarter points for the Ravens: 14
Enter the 1st quarter points for the Steelers: 0
```

```
Enter the 2nd quarter points for the Ravens: 21
Enter the 2nd quarter points for the Steelers: 0
Enter the 3rd quarter points for the Ravens: 3
Enter the 3rd quarter points for the Steelers: 0
Enter the 4th quarter points for the Ravens: 10
Enter the 4th quarter points for the Steelers: 0

Game Summary
------------
Ravens:    14   21    3   10
Steelers:   0    0    0    0

Ravens:    48
Steelers:   0
```

Problem 3

Create a **Sphere** class that will allow you to create Sphere objects. Also create a **SphereTester** program (with a main method).

Your Sphere class should have:

- one field of type double that holds the radius
- a constructor that accepts one argument (the radius' length),
- and the following two methods:
    1. getVolume – a method that computes and returns the sphere's volume
    2. getRadius – a method that returns the radius

Then create a program (i.e., a class that has a main method) called **SphereTester** that prompts the user for the radii of two spheres in meters. The program should display which sphere is larger and by how many cubic meters. If the spheres are the same size, simply display that information. Display only four digits after the decimal point.

Sample run:

```
Enter the radius of a sphere (in meters): 1
Enter the radius of a 2nd sphere (in meters): 2
Sphere 2 is greater than Sphere 1 by 29.3215 cubic meters
```

Sample run:

```
Enter the radius of a sphere (in meters): 3.4
Enter the radius of a 2nd sphere (in meters): 3.4
The spheres are the same size
```

The following program 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;
    }
}
```