1 Exercise

For this lab you may work together but each person will submit their own programs. As usual, there is no file sharing allowed.

In this lab you will be working with arrays of objects that you have created. You will construct a sphere class that will have several methods, including one that will take in another sphere as a parameter and return true if the two spheres collide and false if they do not. You will also construct an array of random spheres and determine the number of collisions and store a list of the spheres that collide.

These calculations lie at the heart of nearly all computer and video games. In most games, especially first-person-shooters, scores are increased when a bullet or laser beam hits (collides) with a target. Also, if the player runs into a wall they do not go through the wall (well usually) but instead bounce off of it or slide along it. Determining what objects hit each other is called *collision detection*. Collision detection is usually taken care of in a game's physics engine that is responsible for most object placement throughout the play of the game. In a physics engine, the objects in a scene are given physical attributes, such as shape, size, velocity, mass, momentum, bounce, frictions, rotational velocity, gravity, and even squishiness. The physics engine uses these attributes and time to determine how the objects interact with each other and where their new positions will be. As a part of that, the engine will determine which objects have collided and adjust their velocities and momentums accordingly. General collision detection is a very complex problem, but for simple objects like spheres (say in a billiards game) it is quite doable.

Create a Sphere class (object) that holds the center of the sphere and its radius. The center of the sphere will be three decimal coordinates (x, y, z) and the radius is just a decimal number. Make sure that the object has a constructor that sets all of the data in the Sphere class and accessor functions that can both get all the data from the object and set all of the data in the object. Have your accessor functions, as well as the constructor, make sure that the radius of the sphere is greater than or equal to 0. Add in methods for finding the volume and surface area of the sphere. Add in a method called `collide` that takes in a Sphere as a parameter and returns true if the calling sphere collides with the input sphere and false if they do not. Testing if two spheres collide is fairly easy, you calculate the distance between the centers of the two spheres and if this is less than or equal to the sum of the radii of the two spheres then the two are colliding. The distance between two points in three dimensions is

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Finally, add a method that will print the sphere data to the screen, that is, the sphere's center, radius, volume and surface area. Make sure that each method of the Sphere class is commented with a description of the method, the input parameters and what they are, and the output of method and what it is, if there is an output to the method.

- Volume of a Sphere: $V = \frac{4}{3}\pi r^3$
- Surface Area of a Sphere: $A = 4\pi r^2$

Write a main program that will ask the user four questions at the start,

```
Input the number of spheres (5-100): 20
Print Sphere Information (Y/N): y
Print Collision List (Y/N): y
Print Maximum Collision List (Y/N): y
```

The first is the number of spheres to create, the second is to print out the sphere information, third to print out a collision list, and a maximum collision list. Do data input checking on all of these inputs. If the number of spheres is not between 5 and 100 the program should ask for the input again until the user types in a number between 5 and 100. Same goes for the character inputs. Here you should take in a string and extract the first character. If the characters are not Y, y, N, or n then the program should ask for the input again. The answers to the last three questions are to be stored as single characters.

The program then creates an array of random spheres, as many a the user specifies. The centers should be randomly generated so that each coordinate is a double between -10 and 10. The radius of the sphere should be a random double between 1 and 3.

The program should then find all the spheres that collide. If it finds a collision the program should store all of the collision information in an array. Really all that needs to be stored here is the number of spheres, so if spheres 4 and 11 collide the array should contain the numbers 4 and 11. One question that needs to be answered is how big to make the array to store the list of collisions? We could calculate the largest possible number of collisions and then use this. That would be a lot of overkill and much wasted memory space. We could simply count the number of collisions in one loop, declare the collisions array of that size, and then go through the sphere data again to populate the array with the numbers of the colliding spheres. This way we will know exactly the correct size of the array to store the collision data.

The program should then find all the spheres that have the maximum number of collisions. That is, for each sphere find the number of spheres it collides with, and then find the maximum of these numbers. Furthermore, find all the spheres that collide the maximum number of times. It is very possible that there are several spheres that collide with the same number of other spheres. All of these maximum collision spheres are to be stored in an array.

If the user selects to see the sphere information then the program should print out the center, radius, volume, and surface area of each sphere.

```
Center: (5.07972603700521, 8.382451098049518, 5.986081502426652)
Radius: 1.7061914719633906
Volume: 20.805200717552825
Surface Area: 36.581827525391425
```

If the user selects to see the sphere collision list, then the program should print out the listing of all collisions, as below. The list should list each collision only one time, so if spheres 4 and 11 collide it should list 4 - 11 and not both 4 - 11 and 11 - 4.

```
Number of Collisions: 6
Collision List
1.  0 - 3
2.  4 - 5
3.  4 - 11
4.  8 - 15
5.  8 - 18
6.  13 - 16
```

If the user selects to see the maximum sphere collision list, then the program should print out the listing of all spheres that have the maximum number of collisions, as below.

```
Maximum Number of Collisions: 2
```

```
Spheres with 2 Collisions: 4  8
```

A sample run of the program is below.

```
Input the number of spheres (5-100): 20
Print Sphere Information (Y/N): y
Print Collision List (Y/N): y
Print Maximum Collision List (Y/N): y

Center: (5.07972603700521, 8.382451098049518, 5.986081502426652)
Radius: 1.7061914719633906
Volume: 20.805200717552825
Surface Area: 36.581827525391425

Center: (-6.807076495311934, 3.224892834110493, -5.376386237794126)
Radius: 2.8159353196165737
Volume: 93.53122761664173
Surface Area: 99.64493179059657

Center: (-4.069674646015368, -0.1361157419393635, -9.411059601111408)
Radius: 1.2482215489329997
Volume: 8.146360722650074
Surface Area: 19.579122142892942

Center: (7.8584397744780645, 8.962312545185075, 3.606660565315547)
Radius: 2.6985009167537015
Volume: 82.31070455419187
Surface Area: 91.50714462592627

Center: (3.8234369820430576, -8.872560952838334, -6.749469836089799)
Radius: 2.5649029725879666
Volume: 70.68079601806485
Surface Area: 82.6707249047497

Center: (2.5218882947438015, -8.314814817798702, -8.843475324903586)
Radius: 1.6041130603290268
Volume: 17.28994188763025
Surface Area: 32.33551733083671

Center: (4.371071239265882, 3.7934384233761165, 6.733765855247956)
Radius: 1.768960225727202
Volume: 23.18690665286731
Surface Area: 39.32294177502279

Center: (-8.01732638171239, 9.609345339706167, 6.757231133935978)
Radius: 1.5400336731033553
Volume: 15.29957122988578
Surface Area: 29.80370786124816

Center: (6.485130781149195, -1.9267783344186498, -1.7699428975819504)
Radius: 2.604659519430519
Volume: 74.0187063778793
Surface Area: 85.2534150729183

Center: (-8.993751798266436, 4.844346401888524, 6.032163408544907)
Radius: 1.0900863593451413
Volume: 5.425894246355314
Surface Area: 14.932470807950116

Center: (3.819157135610764, 5.021585936088712, 2.125824596738461)
Radius: 2.7237433489418272
Volume: 84.64224183740575
Surface Area: 93.22711172874187

Center: (7.971059471936552, -7.495571914334911, -8.532269712608933)
Radius: 2.2719502851238715
Volume: 49.123022529014015
Surface Area: 64.86456528207314

Center: (1.655634145962864, 1.1976821406433498, -8.719026754022698)
Radius: 1.3994746961636702
Volume: 11.48110689711196
Surface Area: 24.61160661622116
```

```
Center: (4.156364084387514, 2.7357422182571174, -5.030690280134593)
Radius: 1.6958461189605507
Volume: 20.429038872763154
Surface Area: 36.1395505954601

Center: (-4.120528558260461, -7.000022551039493, 9.70995913921793)
Radius: 1.0303548140475975
Volume: 4.581936044934381
Surface Area: 13.340849139923709

Center: (7.81153700110875, -4.574538631420319, -2.3599102072806595)
Radius: 1.6261756174616138
Volume: 18.013203047226117
Surface Area: 33.23110281657754

Center: (5.678294410032107, 2.9409171790001416, -5.443932127475568)
Radius: 1.285075674592516
Volume: 8.889446444183553
Surface Area: 20.75234934394577

Center: (-2.8525004862398102, -7.641599311287573, 7.17343903474805)
Radius: 1.047678027536257
Volume: 4.816949758177592
Surface Area: 13.793215944898371

Center: (6.610167214492556, -4.87308862789307, 0.24952603282191)
Radius: 1.2402708220204206
Volume: 7.991681915385017
Surface Area: 19.330492438013927

Center: (-8.609106011537959, -3.4132291207487286, -1.7505765859612374)
Radius: 1.337665679585055
Volume: 10.026084555330254
Surface Area: 22.485628602896544

Number of Collisions: 6
Collision List
1.  0 - 3
2.  4 - 5
3.  4 - 11
4.  8 - 15
5.  8 - 18
6.  13 - 16

Maximum Number of Collisions: 2
Spheres with 2 Collisions: 4  8
```

2 Submit

1. The Java code files for main program and the Sphere class.
2. A Word, LibreOffice, or text file containing at least three runs of the program.