CSCI 221, Spring 2017
Programming HW 2, 50 points

Due date, Friday, February 17, 11:59 pm
Leniency 24 hours later (see syllabus for details)

Collaboration policy: This is an assignment to be completed **individually**. General discussion among humans in this class and elsewhere, about how to use Java, how to create zip files, etc. are acceptable. But **no discussion** of how to solve the problem is allowed. You may post general questions, but no code, to piazza. Questions asking for clarification of the specifics of the problem are allowed. **You may NOT share/edit/provide/give/get/view the code of others.**

This assignment will give you practice with writing a client application of one class and implementing a second class. The classes store location information for various places of interest. The target application is something like Google Maps where a user can make requests such as asking for restaurants that are near a certain location.

This assignment is not as large as HW1, so less time should be required to complete it, and it is worth 50, rather than 100 points.

Part 1: Location Data (5 points)

The code you are writing will be more interesting to use if we have a lot of data to work with. Companies like Google have invested significant resources to identify places of interest and to record their name, their address, and their location along with some tags that are relevant for searching. We don't have the resources of a company like Google, but we have a lot of students. So we will use a crowdsourcing approach by having each student provide information for at least 10 places of interest in the general Charleston area. You can include any place that you think a CofC student would be likely to go including, for example, restaurants near the airport or campus or across town.

For each location, we want the following information:

- Name of the place of interest
- Street address (or similar description) for the place (not a complete address, just enough information to find it if you were near that location)
- One or more search tags for this place
- The latitude of this place
- The longitude of this place

For example, suppose you want to include the Aquarium near Harbor Walk in our data file. You know its name and address and can come up with some tags. To find its latitude and longitude, you can go to this web page and search for South Carolina Aquarium:

http://universimmedia.pagesperso-orange.fr/geo/loc.htm

Or you can get the information directly from Google Maps by following the instructions here:

http://www.wikihow.com/Get-Longitude-and-Latitude-from-Google-Maps

You can add whatever search tags you think are appropriate, but Google provides a list of standard tags that would be useful to apply to your entries:

https://developers.google.com/places/documentation/supported_types

You should put the information together on separate lines, as in:

```
South Carolina Aquarium
Aquarium Wharf, Charleston
aquarium, point_of_interest
32.790829
-79.9256987
```

You are to choose at least 5 locations and come up with a similar 5-line entry for each location. The locations should not be private residences and should not include offensive descriptions. Put it all together in a file called **places.txt**. That means your file will have at least 25 lines. You can use any text editor you like, but be sure to save it as plain text. Include a blank line between entries to make it easier to distinguish them while editing.

NOTE: finding data for places.txt is part of the Monday, February 13, in class lab!

Part B: GeoLocationClient.java (5 points)

In this part of the assignment you will write client/driver code to manipulate some GeoLocation objects. The GeoLocation class is being provided to you, so you don't have to write it. You will instead be writing code that constructs and manipulates three GeoLocation objects (the code will be in a file named GeoLocationClient.java)

The popular TV series Breaking Bad made use of geographic location information. The Walter White character buried millions of dollars at a particular location in the desert outside of Albuquerque, New Mexico. He then bought a lottery ticket to help him remember that his stash was buried at a latitude of 34 degrees, 59 minutes, 20 seconds and a longitude of -106 degrees, 36 minutes, 52 seconds. Your client program will compute the distance between Walter's stash and the local FBI building and a local studio known as ABQ Studios (where Breaking Bad was filmed). Walter's stash was supposedly buried in the desert, but from this client program, you'll see that the coordinates they gave on the TV show are really the coordinates of the studio.

Your program is required to produce the following output:

```
the stash is at latitude: 34.988889, longitude: -106.614444
ABQ studio is at latitude: 34.989978, longitude: -106.614357
FBI building is at latitude: 35.131281, longitude: -106.61263
distance in miles between:
    stash/studio = 0.07548768123801672
    stash/fbi    = 9.849836190409732
```

You should use the latitude and longitude values in this log to construct the three GeoLocation objects. You should then print the three objects and call the distanceFrom method twice to get the desired output. Please note that the latitude/longitude information in the first three lines of output has to be produced by calls on the toString method of the GeoLocation class and the values in the final two lines of output have to be produced by calls on the distanceFrom method of GeoLocation.

Part C: PlaceInformation.java (20 points)

For this part of the assignment, you will write a class called PlaceInformation that stores information about a place of interest. It must have private instance variables that stores a place's:

- name – which is a String
- address – String
- tag(s) – String
- latitude and longitude – in a GeoLocation object (not as two doubles)

It must have the following public methods:

```
public PlaceInformation(String name, String address, String tag,
                        double latitude, double longitude)
public String getName()
public String getAddress()
public String getTag()
public String toString()
public GeoLocation getLocation()
public double distanceFrom(GeoLocation spot)
```

The first three "get" methods simply return the values that were provided when the object was constructed. The toString method should return the name followed by the location details inside parentheses like this (latitude: ##.#######, longitude: ##.#######). Although the constructor takes a latitude and longitude, you should store this information inside the PlaceInformation object using a GeoLocation object. The getLocation method should return a reference to this GeoLocation object. Remember that in writing your class, you don't want to include code that appears elsewhere. For example, your GeoLocation object knows how to compute a distance, so you should not be repeating the code for computing distances in your PlaceInformation class. You should instead be asking the GeoLocation object to perform the computation. Be sure to properly encapsulate your object with private fields.

This class is similar to the GeoLocation class, so you can use it as a model for how to write your own class. **The GeoLocation class will also provide a useful example of how to comment your class.** Each method should be commented and the class itself should have a general comment. And, of course, your name must be included in every file.

We are providing a client program called **PlaceInformationClient.java** that can be used to test your class. We are also providing a more sophisticated client program called **SearchNear.java** that will use the data that students are providing about places nearby, although that program is an optional extra that is just for fun. **SearchNear.java** will only run if **places.txt** is available. **GeoCoder.java** looks up place information from Google Maps, and is used by **DistanceFinder.java** that depends on your **PlaceInformation.java** is available.

I've discussed several classes here, but you write only two: GeoLocationClient.java, PlaceInformation.java

Submit a zip file entitled `<YourLastName>HW2.zip`, that when unzipped reveals a folder with the name `<YourLastName>HW2`, and in that folder we should find the following files: `places.txt`, `GeoLocationClient.java`, `PlaceInformation.java`, `GeoLocation.java`, `PlaceLocationClient.java`

Note – to create such a zip folder:

1. Create an empty folder with the name <lastname>HW2
2. Move the provided files (GeoLocation.java, PlaceLocationClient.java) and your own files (GeoLocationClient.java, PlaceInformation.java and places.txt) files into it.(NOTE: these file must be named exactly as shown and the class names must match the file names, i.e. GeoLocationClient.java stores class GeoLocationClient).
3. Zip/compress it into <lastname>HW2.zip
4. Upload to OAKS.

To test if you have created a proper submission file, assuming your last name is Nero:

1. Unzip NeroHW2.zip
2. You should now see a folder named NeroHW2
3. Look in NeroHW2 and see the java and text files mentioned in 2 above.
4. If 1, 2, & 3 succeeded, success! Submit NeroHW1.zip

Grading Rubric

10 Points	Style: Comments and Indentation
40 Points	Functionality: • Program compiles (5) • Program runs (5) • Programs produce correct results (30)

- If the submitted program does not compile: 0 to 10 points
- If the submitted program compiles but does not run: 5 to 15 points
- If the submitted program compiles and runs: 10 to 40 points
- If the submitted program compiles, runs, and produces correct output: 40 of 40 points

The correctness of your program will be evaluated using test cases developed by the instructor.

Late assignments will not be accepted – no exceptions (please do not email me your assignment after the due date, I will ignore it).

Please feel free to setup an appointment or drop by during office hours to discuss the assigned problem. I'll be more than happy to listen to your approach and make suggestions.

NOTE: I do not promise to be available to answer questions after 3:30pm on Fridays or over the weekend.