Assignment GPS including jUnit Tests CSIS-1410

Learning Objectives:

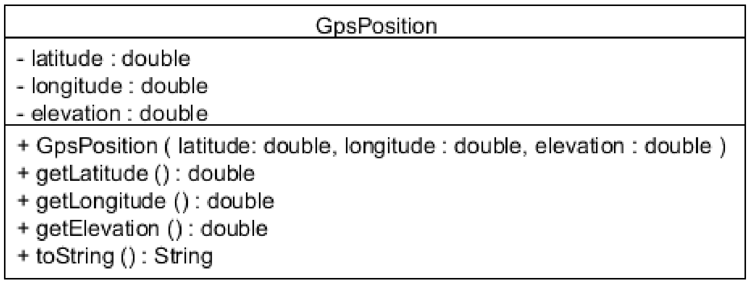
- Practice Composition
- Review declaring classes based on UML class diagrams
- Use jUnit tests to test and troubleshoot your code

Description:

- Create a new Java project called "1410 Gps". It includes a package named `gps`.
- Inside the package `gps` implement the classes `Gps` and `GpsPosition` as described below.
- Include doc comments for each of the public methods, for the constructors, and for the classes themselves.
- Use the provided jUnit tests to test and troubleshoot both classes .
- Feel free to add a main method at the end of the classes `Gps` and `GpsPosition` to demonstrate the use of the class members. For grading purposes the main methods will be ignored.

Class **GpsPosition:**

Class GpsPosition represents a position based on a given longitude, latitude, and elevation (in meters).
Declare class `GpsPosition` based on the UML class diagram below. Notice that there are no setters. The class is immutable. This means the state of a GpsPosition object can no longer be changed once it has been created.
Do not add or remove any public methods or constructors (except for main).
There should also be NO print statements in class `GpsPosition`.



```
GpsPosition
- latitude : double
- longitude : double
- elevation : double
+ GpsPosition ( latitude: double, longitude : double, elevation : double )
+ getLatitude () : double
+ getLongitude () : double
+ getElevation () : double
+ toString () : String
```

The getters expose the field values as usual.

Constructor:
The constructor should implement input validation for .
If the constructor receives an invalid argument, an `IllegalArgumentException` should be thrown.
Like this: `throw new IllegalArgumentException("Invalid latitude and/or longitude");`

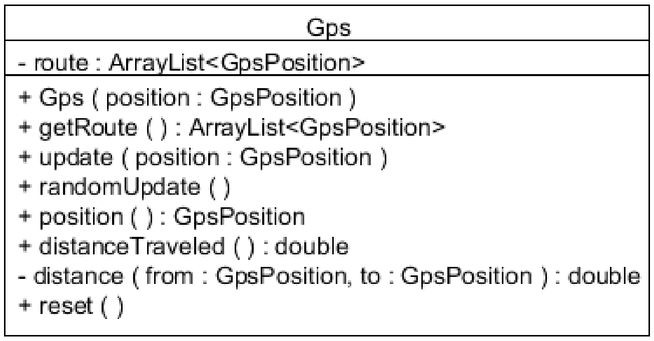
Method `toString`:

The method `toString` should return a String that includes the latitude followed by a comma, a space, the longitude, a space, and then the elevation in parenthesis – like this except with the numeric values:
`latitude, longitude (elevation)`
Both latitude and longitude should display 6 digits after the decimal point while the elevation should display only one digit after the decimal point. Example: `38.573645, -109.546389 (1227.1)`

Use the jUnit tests to test and troubleshoot `GpsPosition` before moving on to the next class.

Class **Gps:**

Declare class Gps based on the class diagram below. No public methods or constructors should be added (except for main). There should be NO print statements in this class.



```
Gps
- route : ArrayList<GpsPosition>
+ Gps ( position : GpsPosition )
+ getRoute ( ) : ArrayList<GpsPosition>
+ update ( position : GpsPosition )
+ randomUpdate ( )
+ position ( ) : GpsPosition
+ distanceTraveled ( ) : double
- distance ( from : GpsPosition, to : GpsPosition ) : double
+ reset ( )
```

The getter exposes the field value as usual.

Constructor:
Use the parameter `postition` to initialize the field `route`. Notice that the argument and the field have different types. When a new Gps is created, the ArrayList should include exactly one element: the start position. You can think of it as the position where the gps is located at that given time.

Method `update`:

The method `update` simulates that the GPS position has changed. The user provides the new position. When the method `update` is called, the new position needs to be added to the route.

Method `randomUpdate`:

Method randomUpdate also updates the position. This time no value is provided by the user. Instead, we simulate the movement of the GPS by calculating a random position within a limited range of the current position.
Calculate the latitude by adding a random numbers from the range [-0.5, 0.5) to the latitude of the current position.
Then calculate the longitude by adding another random number form the same range to the longitude of the current position. Assume the elevation remains unchanged.
When the method `randomUpdate` is called, the newly position needs to be added to the route.

Method `position`:
This method returns the current position. The current position is the last element of the route.

Method `distanceTraveled`:
This method calculates the distance that has been traveled by calculating the sum of all the distances between the positions that are listed in the route – from the original start position all the way to the current position. The distance should be measured in **km**. In order to calculate the distance between two gps positions call the method `distance`.

Method `distance`:
This method should return the distance between two gps positions in **km.**
Notice that minus in the UML class diagram. It indicates that this method should be declared private.
Calculate the distance by using the Haversine formula as shown in this algorithm. You'll need to make two important adaptations though: the different parameter list (as specified in the UML diagram) and the unit (**km** instead of m)
Caveat: remember that the elevation is measured in m

Method `reset`:
This method resets the `route` to the current position, which is the last element of the list `route`.
After a reset, the `route` should include only one element: the current position.

Use the jUnit tests to test and troubleshoot class `Gps` before moving on to the next class.

Video:

Create a screen recording (about 2-3 min) similar to the one you did for the first assignment.
Start with your name. Have the editor pane wide open. Show the code of both classes by calmly scrolling down the code from top to bottom. Point out any known problems with the code.
When done compile and run the unit tests.

Turning in :

Submit both the video AND a jar file that includes the source code