COMP125 2014: Assignment 2 - Handling GPS Tracks

Due: **Week 9: 5pm, Friday 17th October 2014**

Updates

- Clarified elevationGain and distance calculation for track.
- Corrected the return type of `toString` for Track and TrackLog to `String`.
- Removed mention of the setters for the Waypoint class. These are not required.
- Added a comment on the `add` method of the Track class relating to the updating of distance and elevation gain values.
- Added the value for the radius of the Earth.

For this assignment you will write software to handle collections of GPS tracks of the kind that is generated by portable fitness trackers. A GPS track consists of a series of measurements of position. Your software will read these tracks and calculate summary statistics such as distance travelled and average speed. You will also write code to handle a collection of tracks for a user that will enable them to see their longest trips or those with the most elevation gain.

This document sets out the main requirements for the assignment but more detail is given in some of the unit tests that you are supplied with. For this assignment you will be given most of the required tests, but you will be asked to write some tests yourself. You will be assesed based on your code passing our tests and on the quality of your code, tests and documentation.

The work for this assignment is split into three parts corresponding to the three classes that we will use. Each part builds on the earlier parts and you should work on them in order, passing the tests for one class before you start on the next. The `Waypoint` class is the simplest as it just represents a point on the globe and the methods involve calculations on latitude and longitude. The `Track` class is next and is probably the most complicated part of the assignment; writing this involves reading CSV files and managing an array of object instances. Finally the `TrackLog` class manages an array of instances of Track and requires that you implement some sort methods.

A starter pack is available at
`https://github.com/stevecassidy/comp125-2014-gpstracks/archive/master.zip`.

All queries to Steve.Cassidy@mq.edu.au or via the Assignment Discussion Forum on iLearn.

The `Waypoint` Class

A waypoint is a single measurement taken by the GPS system, it records the latitude, longitude and elevation along with a timestamp. The Waypoint class implements a simple interface for storing this information and performing a few operations on it. The Waypoint class provides the following interface:

- `public Waypoint(final String wstring) throws GPSException` the class constructor takes a string that represents a single waypoint in Comma Separated Values (csv). The string will contain the fields: timestamp, latitude, longitude and elevation separated by a comma. For example: `"2014-08-22T20:18:44Z,-33.7940910,151.0470420,-2.2"`. The constructor will throw a GPSException if there is a problem with the format of the string argument.
- `public String getTimestamp()`, getter for the timestamp value for this waypoint. Timestamps are stored as a string in ISO format `"yyyy-MM-dd'T'HH:mm:ss'Z'"`, eg `"2014-08-22T20:18:23Z"`. *Note that for this assignment you can treat timestamps as a string internally, we will not need to do anything other than sort on timestamps this time.*
- `public double getLatitude()` getter for the latitude value for this waypoint (*setter is not required*).
- `public double getLongitude()` getter for the longitude value for this waypoint (*setter is not required*).

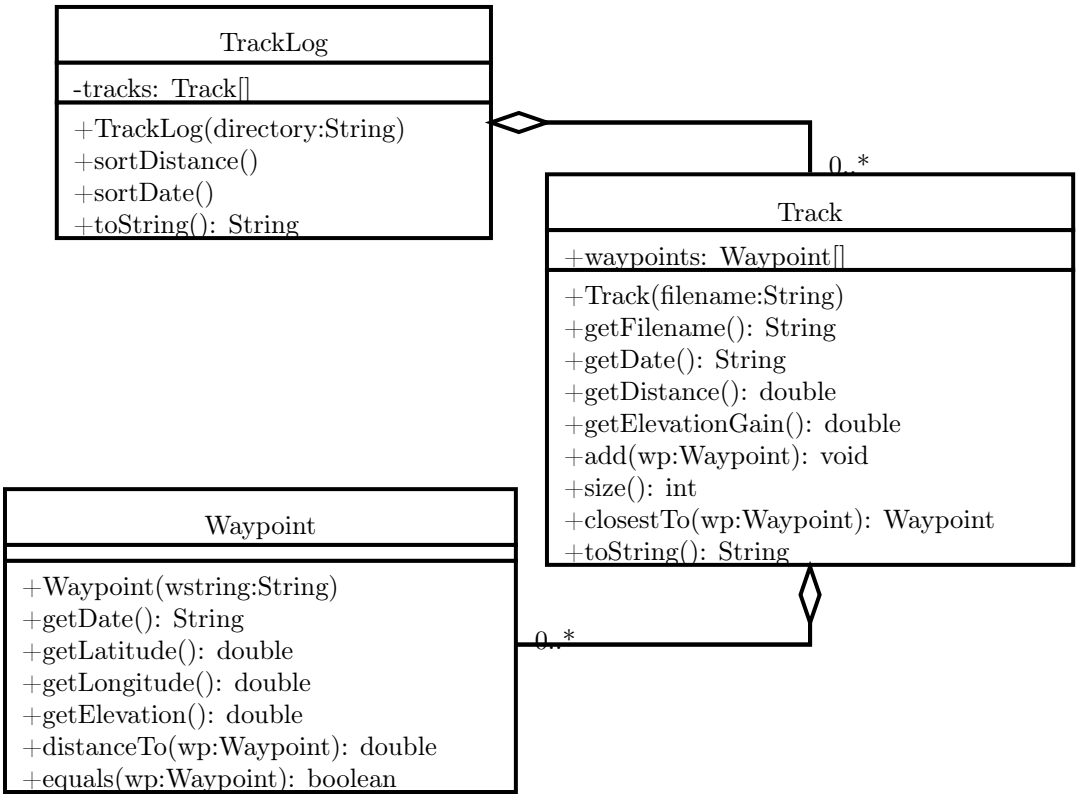

Figure 1: UML diagram for the Waypoint, Track and Tracklog classes. A TrackLog contains zero or more Tracks, and a Track contains zero or more Waypoints.

- `public double getElevation()` getter for the elevation value for this waypoint (*setter is not required*).
- `public double distanceTo(Waypoint wp2)` Calculate the distance to a second waypoint. Distance is calculated using the Haversine formula for great circle distance between two points on the earth's surface. Details are provided below.

The `Track` Class

The Track class represents a single trip recorded by the GPS. It holds an array of Waypoint instances representing individual position measurements along the track. It provides summary information about the track such as the distance and overall elevation gain.

Track instances are constructed from a file containing a series of waypoints; the distance and elevation gain need to be calculated from the waypoints.

For this assignment, the array of Waypoint instances is limited to be of size less than or equal to 1000. If a file contains more than 1000 waypoints, the later ones are silently ignored. *Note that this is not good practice, we'll extend the class to be better behaved in the final assignment.*

The Track class provides the following interface:

- `public Track(String filename) throws IOException, GPSException` this class constructor creates a Track instance by reading a file containing waypoints in CSV format. The values of distance and elevation gain are calculated from the waypoints in the track. The constructor will throw an IOException if there is any problem reading a file (eg. file is not found) and a GPSException if there is a problem with the file format.

- `public String getTimestamp()` return the timestamp for this track, the timestamp is the same as the timestamp for the first waypoint in the track. The timestamp format is the same as that used in the Waypoint class.
- `public double getDistance()` get the distance for this track. The distance can't be modified once the track is created. **Note: the distance is the sum of the distances between pairs of waypoints along the track.**
- `public double getElevationGain()` get the elevation gain for this track. The elevation gain can't be modified once the track is created. **Note: Elevation gain is defined as the positive change in elevation along the track - the sum of any positive elevation difference between pairs of neighbouring waypoints.**
- `public String getFilename()` get the filename that the track was read from.
- `public void add(Waypoint wp)` Add a new waypoint to the end of the track. **NOTE: to maintain the integrity of the class, this method should update the distance and elevationGain measures. This has not been included in the tests and you will not be penalised for not implementing this, but you may want to do so anyway.**
- `public int size()` returns the number of waypoints stored in the track.
- `public Waypoint closestTo(Waypoint wp)` returns the waypoint in the track that is closest to the given waypoint.
- `public String toString()` return a string representation of the track suitable for printing. The string should contain all of the summary statistics for the track.

The `TrackLog` Class

A TrackLog is a collection of tracks. We will create a tracklog from a collection of CSV files. This class provides methods for sorting the list of tracks in different ways and displaying summary statistics about the tracks to the user.

The TrackLog class stores the individual Track instances in an array. For this assignment, the array of Track instances is limited to be of size less than or equal to 20. If a directory contains more than 20 tracks, the later ones are silently ignored. *Note that this is not good practice, we'll extend the class to be better behaved in the final assignment.*

The TrackLog class provides the following interface:

- `public TrackLog(String directory)` The class constructor scans a directory and creates tracks for each CSV file found in that directory. *An implementation of this method is provided for you, it assumes that the `Track` class has been properly implemented.*
- `public int size()` returns the number of tracks stored in this track log.
- `public void add(Track tr)` adds a new track to the end of the track log.
- `public Track get(int index)` returns the track in the position given by the index or null if the index is out of bounds.
- `public void sortDistance()` sort the tracks in the TrackLog based on the distance of each track. The sort modifies the track array and must be a stable sort. **You may not use the built in sort implementation.**
- `public void sortTimestamp()` sort the tracks in the TrackLog based on the timestamp of each track. The sort modifies the track array and must be a stable sort. **You may not use the built in sort implementation.**
- `public String toString()` return a string representation of the list of tracks suitable for printing. The list should contain one line per track containing all of the summary statistics for that track. The list will be in the order that the tracks are currently stored.

The `GPSException` Class

If an error condition is detected in this module an exception of this type is raised. An implementation of this class is is included in the starter pack.

Sample GPS Files

You are provided with a collection of GPS track files in the appropriate CSV format. These are generated by converting a GPX format file exported from the website Strava into our CSV format. The files represent bike rides ridden by me, mostly around Sydney but with a couple a bit further afield.

The Haversine Formula

The usual way to find the distance between two points if you have their X and Y coordinates would be to use Pythagoras (square root of the sum of the squares). When we're dealing with latitude and longitude it gets a bit more complex since we are on the surface of a sphere (well, almost a sphere). The shortest distance between two points on the surface of the Earth is defined by the *great circle distance* - that is we look at the path between the two points that if continued would go all the way around the globe. To work this out is complex but there is an approximation that works well called the Haversine Formula. I've found the website at `http://andrew.hedges.name/experiments/haversine/` to be a useful reference. It has a Javascript implementation that gives you the formula in a form that is almost right for Java and a web form so that you can check the distance between two points yourself. Here's that implementation:

```
1        dlon = lon2 - lon1
2        dlat = lat2 - lat1
3        a = (sin(dlat/2))^2 + cos(lat1) * cos(lat2) * (sin(dlon/2))^2
4        c = 2 * atan2( sqrt(a), sqrt(1-a) )
5        d = R * c (where R is the radius of the Earth)
```

Take the radius of the Earth to be 6373.0km, as per the page linked above.

To translate this into Java you need to know that the trigonometric functions are in the Math library (eg. `Math.sin`) and that you need to convert the latitude and longitude angles to radians using `Math.toRadians` before calculating the sines and cosines.

Documentation/Comments

The code you are given in outline form contains no comments. You should fill in the JavaDoc style comments before each method to provide useful documentation to anyone who will use your implementation. Javadoc is a style of writing comments that enables documentation for your code to be automatically generated. It encourages you to write a descriptive comment at the start of every method and fully describe the formal parameters. The Wikipedia page (`http://en.wikipedia.org/wiki/Javadoc`) gives enough of an overview for you to get the idea. In this assignment, you should include the description as well as the @param and @return comments for each method.

You should also include comments in your code where appropriate, for example to explain what a complex for loop or conditional test is doing. Try to balance providing useful comments with stating the obvious, we don't want to see comments like "Add one to x".

Unit Tests

You are provided with a set of JUnit tests for all classes which test the provided methods and those you are asked to write above. Part of the mark for this assignment will be based on your code passing these tests.

You are also required to complete the tests for the `TrackLog` class to test the three methods that you must write. In grading your assignment, we will run our own set of tests against your implementation. Your tests will be checked manually for quality and completeness.

Using Your Classes

Your final task, once you have passed all of the tests that are provided, is to write a simple application that makes use of the classes that you have implemented. This means that you need to define a `public static void main(String[] args)` method either in the TrackLog class or in a new class that you define. Your application should:

- prompt the user to enter a directory name
- generate a track log from the files in that directory
- print out summary statistics for the tracks in timestamp order
- print out summary statistics for the tracks in distance order

Submitting Your Work

You should submit an exported version of your Eclipse project. Select Export from the file menu, then choose General > Archive File. This should create a zip file containing your source code (it should have the same structure as the zip file you are given to start the project).

Upload the zip file to iLearn in the **Assignment 2 Submission** activity.

Marking

You will be marked based on the number of tests that you are able to pass and on the quality of your code and the documentation and comments that you provide. The assignment is worth 10% of the overall mark for the course but will be marked out of 100 and scaled. The marks are allocated as follows:

- (10 marks) Your implementation of the Waypoint class passes our JUnit tests.
- (25 marks) Your implementation of the Track class passess our JUnit tests.
- (25 marks) Your implementation of the TrackLog class passes our JUnit tests.
- (5 marks) Your tests for the TrackLog class are complete and effective tests of the required functionality
- (10 marks) The completeness, quality and readability of the documentation you provide in your code including JavaDoc documentation strings and inline comments.
- (20 marks) The readability and quality of your Java code and your implementation choices for the different required methods.
- (5 marks) Your implementation of the requirements under Using Your Classes above.