Exercise 1: Class for GPS data points

In this assignment, a class GPSPoint.java will be implemented to be able to represent GPS data points.

The idea is that a GPS point should be represented as an object of the GPSPoint class and the program should have such an object for each GPS point. For each GPS point, we need to store time, latitude, longitude and altitude.

a) Object variables and constructor

Look at the start code for the class GPSPoint.java in the package no.hvl.dat100ptc.askgave1. Extend the start code for the class so that the class has the following object variables:

hour (integer) which indicates the time in seconds

latitude (decimal number) which indicates latitude

longitude (decimal number) indicating longitude

elevation (decimal number) which indicates height in meters

which all must be private ie only visible within the class.

Furthermore, the class must have a constructor

public GPSPoint (int time, double latitude, double longitude, double elevation)

which can add value to all object variables.

Test the implementation by running the tests in the test class GPSPointeTester.java

b) Retrieve / set methods

Complete the implementation of get / set methods and test them with the device tests.

c) Representation as a text string

Complete the implementation of the toString () method, which returns a string representation of a GPSPoint object in the form:

1 (2.0,3.0) 5.0 \ n

where 1 is the time, (2.0,3.0) is (latitude, longitude) and 5.0 is the altitude.

Test the implementation using the device tests.

**Exercise 2 - GPS data reading and reference tables**

This part of the project is about how GPS data can be read from a file and how the GPS points in the file can be represented by using a table with pointers to GPSPoint object.

The GPSDataRFileReader.java class contains ready-made Java code for loading a CVS data file with GPS points in the format explained earlier. There is no need to change this Java code and it is only a little later in the subject that we will consider how to read from - and write to - files in Java.

The focus of this task is on implementing methods in the class GPSData.java that will store loaded GPS data points in the form of a reference table of GPSPoint object and the help class GPSDataConverter.java which contains auxiliary methods for converting the data read into GPSPoint object.

a) GPS data conversion

When data is read from file, information about the GPS point will be represented as strings. For example, the time of a read GPS point will be represented as the string:

"2017-08-13T08: 52: 26.000Z"

Complete implementation of the method

public static int toSeconds (String timestr)

which converts time data as represented above to the number of seconds. For the example above, the 8 hours, 52 minutes and 26 must be converted to seconds and returned.

Hint: Look at the substring method in the String class as well as the parseInt method in the Integer class.

Implement the method

public static GPSPoint convert (String timeStr, String latitudeStr, String longitudeStr, String elevationStr) {

which takes String representations of time, latitude, longitude and altitude, converts these and creates a GPSPoint object with the corresponding data. The method should return a pointer to the GPSPoint object that was created.

If, for example, the method must be called with

convert ("2017-08-13T08: 52: 26.000Z", "60.385390", "5.217217", "61.9")

then the method must return a GPSPoint object where hour is 31946, latitude is 60.385390, longitude is 5.217217 and elevation is 61.9.

b) Reference table with GPS points

The GPS points that are read in from a file and converted must be represented by using a reference table, ie a table where the stored elements are pointers (references) to GPSPoint objects.

\* For this assignment, Chapter 6.4 and Teaching 14 are particularly relevant.

The start of the implementation can be found in the class GPSData.java:

public class GPSData {

private GPSPoint [] gpspoints;

protected int number = 0;

[...]

The object variable gpspoints should be used to point to the reference table of GPS points. The object variable number must be used in connection with. with insertion in the table to keep control of where (ie at which position / index) the next point is to be inserted.

The variable number will at any time indicate how many GPS points are inserted in the table. An index where no item is inserted will have the value zero (a zero pointer). The object variable number has the modifier protected to make it easier to test the class. The tests for the class can be found in the class GPSDataTester.java.

Complete the implementation of the following methods:

public GPSData (int n) which is a constructor for the class. The designer must create a table of GPS points with the size given by the parameter n and set the number equal to 0 (since the first element must be in position 0).

protected boolean insertGPS (GPSPoint gpspoint) which inserts the GPS point gpspoint in the gpspoints table at the position specified by the object variable number. Furthermore, the method must increment the number so that the next point enters the next position. The method should only insert gpspoint if there is space in the table, ie if the number is strictly less than gpspoints.length. The method should return true if the point was inserted and false otherwise.

public boolean insert (String time, String latitude, String longitude, String elevation) that takes GPS point data in string representation and inserts a corresponding GPSPoint object in the gpspoints table. Tips & Warnings The method should convert data, create a new GPSPoint object, and use the insertGPS method above.

public void print () to print GPS data contained in the gpspoints table in the following form

====== Converted GPS Data - START ======

1 (1.0,2.0) 3.0

2 (4.0,5.0) 6.0

3 (7.0,8.0) 9.0

====== Converted GPS Data - END ======

Tips & Warnings Use the loop and toString method on GPSPoint object

The methods already implemented in the GPSDataFileReader.java class read - line by line - in the GPS data file and store data in the table using the insert method. This means that the points in the gpspoint table correspond to the route represented in the GPS data file.

**Exercise 3 GPS auxiliary methods**

In this problem, some help methods will be implemented in the class GPSUtils.java which we will use later in the project to make calculations on the GPS data points that are read in and stored via the methods and classes implemented in problem 2.

The task is to complete the implementation of the methods below in the class GPSUtils.java and use the device tests in the class GPSUtilsTester.java to continuously test the code.

For this task, Chapters 5.1-5.3, 5.5-5.6 in the Java book as well as Teaching 8 and 9 are particularly relevant

a)

Implement the method

double findMin (double [] da)

which finds the smallest number in a table with floating point numbers. It can be assumed that there is at least one element in the table. Tips & Warnings Look at the implementation of the findMax method that already exists in the class.

b)

Implement the method

public static double [] getLatitudes (GPSPoint [] gpspoints)

which takes a table with GPS points as a parameter and returns a table of decimal numbers containing the latitudes of the GPS points.

Hint: the method should first create a table of decimal numbers with the same length as gpspoint and then copy the individual latitudes over to the new table. Remember that the getLatitudes method of a GPSPoint object can be used to read the latitude of an object.

c)

Implement the method

public static double [] getLongitudes (GPSPoint [] gpspoints)

which is similar to the getLatitudes method above but for longitudes.

d)

Implement the method

public static double distance (GPSPoint gpspoint1, GPSPoint gpspoint2) {

which calculates the distance d in meters between two GPS points on the globe using the Haversine formula

where R = 6371000 meters is the earth's average radius.

Tips & Warnings The Math class: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Math.html contains the methods needed including atan2 and toRadian's methods .

e)

Implement the method

public static double speed (GPSPoint gpspoint1, GPSPoint gpspoint2) {

as calculations average speed in km / h if you move from the point given by gpspoint1 to the point gpspoint2 on the number of seconds given with the parameter secs.

Tips & Warnings Use the distance from d) method as well as the get method (s) on GPSPoint object.

f)

Implement the method

public static String formatTime (int secs)

which returns a string in the format hh: mm: ss where the time in seconds from midnight is given by the parameter secs. In the string of the format hh: mm: ss is hh is the number of hours, mm is the number of minutes and ss is the number of seconds. Furthermore, the method should add spaces before the time so that the total length of the string is 10. Hint: look at the format method in the String class.`

g)

Implement the method

public static String formatDouble (double d)

which rounds a floating point number to two decimal places, inserts the result into a string and fills in the space at the front of the string so that the length of the string becomes 10.

**Exercise 4: GPS-based statistics**

In the class GPSComputer.java, methods will be implemented that calculate statistics (key figures) based on the entered and converted GPS data.

For this task, task 4b) (https://github.com/dat100hib/dat100public/blob/master/programmering/jplab7/JP7.md#b) from programming lab 7 is particularly relevant.

General hint: In connection with the implementation, you should think about whether there are already methods from previous tasks or this sub-task that can be used in the solution. This especially applies to methods from the class GPSUtils.java?

The GPSComputer.java class contains a table

private GPSPoint [] gpspoints;

with (already) converted GPS data and the methods implemented should use data from this table to make calculations. The table will contain the GPS points that make up the route, ie the points that one has cycled through.

The test class for this task is GPSComputerTester.java.

a)

Implement the method

public double totalDistance ()

which calculates the total distance on the route that the GPS data in the gpspoints table indicates. Ie. the method must add the distance (distances) between the points that make up the route.

b)

Implement the method

public double totalElevation ()

which calculates the total number of altitude meters on the route. Remember to only count altitude meters between two points if you move upwards.

c)

Implement the method

public int totalTime ()

which will calculate the total time it has taken to cycle the route corresponding to the entered GPS data.

d)

Implement the method

public double [] speeds ()

which should return a table of average velocities between each of the points we have moved between. Ie. first entry in the table should be the speed we moved at between point 0 and point 1, second input speed between point 1 and 2 etc. If the number of GPS data points is N then the length of the table returned will be N-1.

e)

Implement the method

public double maxSpeed ​​()

which returns the greatest speed we have moved with between two points on the route.

f)

Implement the method

public double averageSpeed ​​()

which returns the average speed we have moved with the total set for the entire route.

g)

Implement the method

public double kcal (double weight, int secs, double speed)

which calculates / estimates how much energy has been burned given the weight of the person and the time in seconds that the person has moved at that speed.

In order to be able to estimate energy metabolism in kilo-calories (kcal), we must first find MET (Metabolic Equivalent of Task) which is a physiological measure of how many kcal we burn per kilo of body weight per hour at a given activity. MET depends on the type of activity and intensity. For cycling [http://coachlevi.com/health/calories-burned-bicycling/] it is given in the table below where speed is given in miles per hour (mps):

Speed ​​MET

<10 mph 4.0

10-12 mph 6.0

12-14 mph 8.0

14-16 mph 10.0

16-20 mph 12.0

> 20 mph 16.0

Speed ​​in km / h can be converted to mph by walking by a factor of 0.62. MET will also depend on ex. slope percentage (whether it goes up or down and how much) but we will ignore that here.

h)

Implement the method

public double totalKcal (double weight)

which calculates the total amount of energy burned on the route.

in)

Implement the method

public void displayStatistics ()

which prints the statistics calculated by the methods in the class. The format of the printout should be as follows:

============================================

Total Time: 00:36:35

Total distance: 13.74 km

Total elevation: 210.60 m

Max speed: 47.98 km / h

Average speed: 22.54 km / h

Energy: 744.40 kcal

=============================================

The tests will call this method with the four log files and the printout will appear in the console in Eclipse.

# Exercise 5 - Visualisering

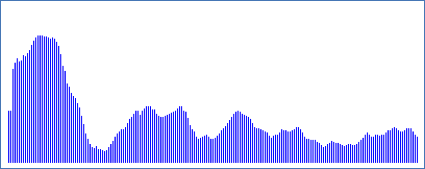
I denne oppgaven skal EasyGraphics-biblioteket brukes til å visualisere høydeprofil, hastighet og rute.

Dokumentasjon for metodene i EasyGraphics-bibliotekt kan finnes her <https://dbsys.info/programmering/easygraphics/javadoc/index.html>

**a) Høydeprofil**

I denne oppgaven skal høyde-kurven for ruten gitt ved GPS datapunktene visualieres.

For GPS datafilen medium.log skal visualiseringen se ut som nedenfor der høyden på en vertikal linje svarer til høyden i GPS datapunktet.

[](https://github.com/dat100hib/dat100-prosjekt-testing-2020/blob/master/docs/assets/markdown-img-paste-20180909115303289.png)

I klassen showProfile.java finnes allerede en main-metode som setter opp et vindu som kan brukes til å tegne høydeprofilen og som ber om navn på den datafil som skal visualiseres (short, medium, long,vm).

Klassen sørger allerede for å lese inn data fra GPS datafilen ved oppstart og lagre GPS data i en tabell gpspoints med GPS punkter

private GPSPoint[] gpspoints;

Implementer metoden

showHeightProfile(int ybase)

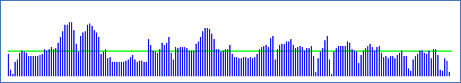
som tegner høydeprofilen der parameteren ybase angir hvor på y-aksen bunnen av en søylene skal starte.

For å gjøre oppgaven enklere kan det antas at hvert punkt (pixel) i vinduet svarer til en høyde-meter. Eventuelt negative høyder skal ignoreres – dvs. behandles som om de hadde verdien 0.

**Hint:** Bruk en løkke for å iterere igjennom alle punktene og oppdater start (x,ybase)-punkt og slutt (x,y2)-punkt for linjer som kan tegnes med drawLine-metoden i EasyGraphics. Den symbolske konstanten MARGIN i klassen angir hvor på x-aksen den først vertikale linjen skal tegnes.

**b) Hastighet**

I denne oppgaven skal hastigheten der blev kjørt med i løpet av ruten visualiseres. For GPS datafilen medium.log skal visualiseringen se slik ut

[](https://github.com/dat100hib/dat100-prosjekt-testing-2020/blob/master/docs/assets/markdown-img-paste-20180909120055723.png)

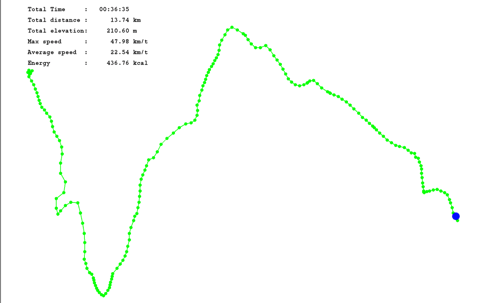
der denne grønne linjen indikerer gjennomsnittshastigheten for hele ruten.

Ferdiggjør implementasjonen av metoden showSpeedProfile i klassen ShowSpeed.java

Der finnes allerede en main-metode i klassen som setter opp vindu og som kaller metoden showSpeedProfile. GPS data blir automatisk lest inn i gpspoints-tabell med samme navn som for høydeprofil ovenfor.

**c) Sykkelruten**

I denne oppgaven ruten visualieres på et kart og til slutt skrive ut statistikk (nøkkeltall) om sykkelturen i øverste venstre hjørne. Et eksempel er vist nedenfor for log filen medium.log

[](https://github.com/dat100hib/dat100-prosjekt-testing-2020/blob/master/docs/assets/markdown-img-paste-20180909120229747.png)

der y-aksen svarer til breddegrader og x-aksen svarer til lengdegrader.

Ferdiggjør implementasjonen av metodene i klassen ShowRoute.java. Der finnes allerede en main-metode i klassen som setter opp vindu og som kaller de tre metodene ShowRouteMap, ShowStatistics og PlayRoute. Startkoden oppretter også et GPSComputer-objekt som kan brukes til å beregne nøkkeltall med.

Implementer metoden

public double ystep()

som beregner hvor mange punkter (pixels) en breddegrad skal svare til for at vi kan tegne alle GPS datapunkter innen for et tegneområde på skjermen med et antall punkter i y-retningen som er gitt ved konstanten MAPYSIZE.

**Hint:** se implementasjonen av metoden xstep(). Vi antar her at jorden er flat dvs. en lengde og en breddegrad svarer til samme avstand uansett hvor vi befinner oss. Al den stund vi ikke sykler over veldig lange avstander er det en rimelig antagelse.

Implementer metoden

showRouteMap(int ybase)

som tegner punkter i vinduet svarende til de (lengdegrad,breddegrad) posisjoner som finnes i GPS datafilen. Parameteren ybase angir det sted på y-aksen som skal svare til den minste breddegrad som finnes i datafilen.

Implementer metoden

public void showStatistics()

som viser statistikk fra sykkelturen i øverste venstre hjørne (se bildet først i oppgaven ovenfor).

**Hint:** Easygraphics-metoden drawString kan brukes til å "tegne" en streng i vinduet. Metodene på objektet gspcomputer inne i klassen kan brukes til å finne de nøkkeltall som skal vises som del av statistikken.