Software Construction 2 (IE-B2-SO2)

Lab Exercise 1:
Geographic Coordinates

HAW Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

1 General Notes

1.1 Primary Learning Objectives

- Simple classes with variables and methods
- Executable classes
- Static elements

1.2 Preparation (Read carefully!)

- Work through the related slide sets provided for the lecture.
- Try to solve the assignments *before* the lab.
- Print the document "*Checklist Software Quality*" and bring it to every lab.
- To achieve the best learning effects, try to solve the assignments by *yourself*.
- Interact with your fellow group members or others to clarify questions or create understanding (e.g., related theory, methods, assignments).
- For each lab, create *one* Java project, that is presented by your group to be approved

1.3 Successful Participation (Read carefully!)

- You must successfully participate in *all* laboratory sessions to participate in the exam.
- You must be *punctual* and *present* during *all* laboratory sessions.
- You must work in teams of *three* students, not alone.
- Make sure all items on the checklist "Software Quality" are fulfilled *before* presenting your code.
- We will assess you *individually*, not your team:
  - You must know the details of each program (i.e., no split-up of the work in a team).
  - You must be able to explain all related theory, concepts, methods, and such.
  - You must convince us that you have achieved the learning objectives.

2 Geographic Coordinates and Distances

2.1 Latitude and Longitude

Geographic coordinates $g = (lat, lon)$ are described by their *latitude* and *longitude*.

- Latitudes are parallel to the equator. The range of values includes -90° ("southern latitude") at the South Pole to 90° ("northern latitude") at the North Pole.
- Longitudes run through the North Pole and the South Pole. The range of values includes -180° ("west longitude") to 180° ("east longitude"). The so-called zero meridian (longitude 0°) passes through the town of Greenwich in England.

What	*Identifier*	*Course*	*Corresponds*	*Value Range*
latitude	*lat*	parallel to the equator	y-coordinate	[-90°, 90°]
longitude	*lon*	through North and South Pole	x-coordinate	[-180°, 180°]

Notes:

- Note that the x- and y-components are reversed compared to conventional coordinates.
- In *Google Maps,* you get the geographic coordinates by clicking on a location.

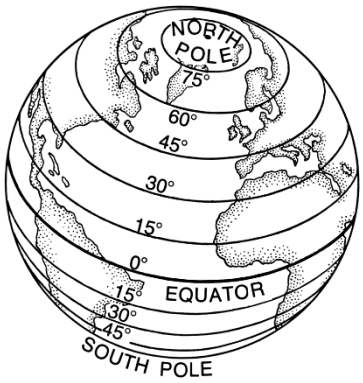


Figure 1: latitudes (Wikipedia[1])

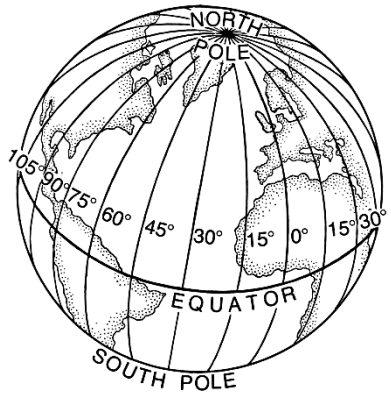


Figure 2: longitudes (Wikipedia[2])

2.2 Local Distance Calculation

In the following, we want to determine the distance between two places on the earth's surface. If two places are relatively close together, one can approximately neglect the curvature of the earth:

1. Determine the distances using of latitude and longitude in km.
2. From this, calculate the direct distance through the theorem of Pythagoras.

[1] https://commons.wikimedia.org/wiki/File:Latitude_(PSF).png (public domain)
[2] https://commons.wikimedia.org/wiki/File:Longitude_(PSF).png (public domain)

Dividing the circumference of the earth in 360°, so 1° corresponds to about 111.3 km. Since the distance between neighboring latitudes is the same everywhere on the earth (Fig. 1), a difference of 1° latitude corresponds to 111.3 km. The distance between two latitudes lat_1 and lat_2 in km is therefore:

$$\Delta y = 111{,}3 \cdot |lat_1 - lat_2|$$

The distance between neighboring longitudes depends on the latitude of the places. At the equator (0° latitude), the distance is 111.3 km. Towards the poles, the longitudes converge and the distance becomes smaller (Fig. 2). At the north and south poles, the longitudes intersect, so there corresponds to the distance 0 km. This is expressed in the following formula by the cosine giving 0 at the equator (0° latitude) and 0 at the poles (±90° latitude). The argument used is the mean of the latitudes of both places:

$$\Delta x = 111{,}3 \cdot \cos\left(\frac{lat_1 + lat_2}{2}\right) \cdot |lon_1 - lon_2|$$

Overall, the distance *d* in km between two geographic locations g_1 and g_2 is:

$$d = \sqrt{\Delta x^2 + \Delta y^2}$$

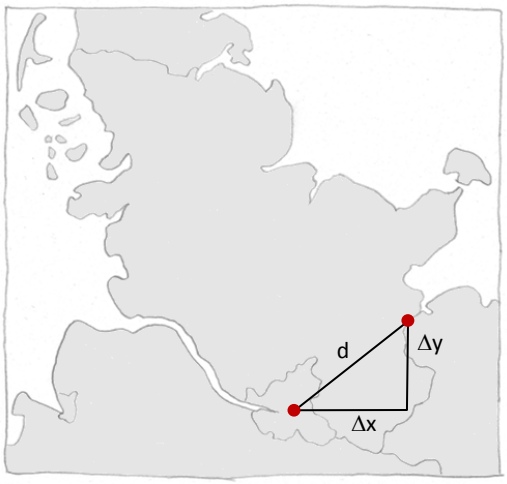


Abbildung 3: Entfernung zwischen Hamburg und Lübeck

2.3 Distance Calculation on the Globe

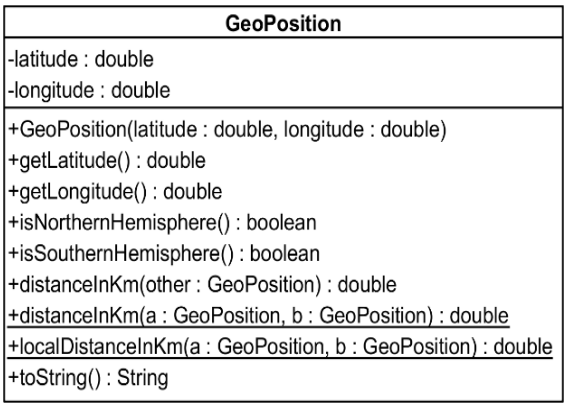
For a more precise calculation of the distance between two places on the earth's surface, the following formula is given without derivation[3]. Here, the factor 6378,388 km corresponds to the earth's radius.

$$d = 6378{,}388 \cdot \arccos(\sin lat_1 \cdot \sin lat_2 + \cos lat_1 \cdot \cos lat_2 \cdot \cos(lon_2 - lon_1))$$

[3] https://www.kompf.de/gps/distcalc.html

3 Class *GeoPosition*

There is a class *GeoPosition* to be created that allows to represent geographic coordinates. The class is basically described by the adjacent UML symbol. In the box below the class name, the attributes are given with the respective data type. A minus sign indicates that the modifier *private* is to be used.
The lower section lists the methods with parameter lists and after the colon the data type of their return value. Underlined elements are class methods.



3.1 Eclipse and Project Structure

a) In Eclipse, create a Java project called “Laboratory”, choosing "*Use project folder as root for sources and class files*" under "*Project layout*". Create the package *lab1.geoPosition* in the project.

b) Add the latest *JUnit* library to the project.

c) Add to the package the given test class *TestGeoPosition*. To do this, you can drag the file *TestGeoPosition.java* from the Windows Explorer into the package in the Eclipse Package Explorer using "*drag & drop*".

3.2 Attributes, Constructors and Getter

Requirement 1 — The class has private variables called *latitude* and *longitude* to store latitude and longitude, respectively. Both variables are of type *double*.

Requirement 2 — There is a constructor with two *double* parameters that assigns the passed parameter values to the attributes. (Hint: Use the menu item "*Source / Generate ...*" in Eclipse to have the method generated automatically.)

Requirement 3 — There are getter methods that return the value of *latitude* and *longitude*, respectively. (Also have these methods automatically generated in Eclipse.)

Declarations:

- `public GeoPosition(double latitude, double longitude)`
- `public double getLatitude()`
- `public double getLongitude()`

3.3 Query of the Hemisphere

Requirement 4 — There are methods to query if a geographic location is in the northern or southern hemisphere.

Declarations:

- `public boolean isNorthernHemisphere()`
- `public boolean isSouthernHemisphere()`

3.4 Distances between two Geographic Locations

Requirement 5 — There are two class methods to calculate the distance (linear) between two locations that are passed as parameters. In both cases, the distance is returned in kilometers. The method *localDistanceInKm()* uses for the calculation the approach presented in Section 2.2. The methods *distanceInKm()* uses the more accurate approach presented in Section 2.3.

Requirement 6 — There is a non-static method *distanceInKm()*, that calculates the distance to a location passed as parameter. The calculation follows the approach presented in Section 2.3.

Notes:

- For mathematical functions use the methods provided by the class *Math*.
- Please consider that the trigonometric functions expect angles to be in radian (not in degree).

Declarations:

- `public static double localDistanceInKm(GeoPosition a, GeoPosition b)`
- `public static double distanceInKm(GeoPosition a, GeoPosition b)`
- `public double distanceInKm(GeoPosition other)`

3.5 Method *toString()*

Requirement 7 — The class has a method *toString()* that returns a string formatted as follows: `(<latitude>, <longitude>)`. (Hint: Let the structure of this method be automatically generated by Eclipse.)

Declarations:

- `public String toString()`

3.6 Unit-Tests

Requirement 8 — Ensure that all given Unit-tests are executed properly.

4 Distances

Create an executable class *GeoApp* (i.e. an executable application), that displays in the console the distance from the HAW Hamburg to all locations given by the following table. For determining the distances use both the local and the accurate approach and enter the results in the table. Additionally, add the deviation of the local and the accurate approach.

Location	*Latitude*	*Longitude*	*Distance in km (accurate)*	*Distance in km (local)*	*Deviation in %*
HAW Hamburg	53,557078	10,023109	-	-	-
Eiffel Tower	48,858363	2,294481			
Palma de Mallorca	39,562553	2,661947			
Las Vegas	36,156214	-115,148736			
Copacabana	-22,971177	-43,182543			
Waikiki Beach	21,281004	-157,837456			
Surfer's Paradise	-28,002695	153,431781			

Follow the same procedure to determine the distances between HAW Hamburg and both Poles as well as the equator.

Location	*Latitude*	*Longitude*	*Distance in km (accurate)*	*Distance in km (local)*	*Deviation in %*
North Pole					
Equator					
South Pole					

5 Compliance with Programming Guidelines (Quality)

Ensure that all quality criteria given by the software quality checklist (provided in EMIL) are met.

6 Theory Questions

1. What is the difference between *primitive variables* and *reference variables*?
2. What are the differences between *instance variables*, *object variables,* a*ttributes* and *local variables*?
3. What would happen when you use the keyword *static* for the variable *latitude*?
4. There are two methods called *distanceInKm()*. Explain the difference.