Software Construction 2 (IE-B2-SO2)

Lab Exercise 2:
Geographic Distances (Routes)

HAW Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

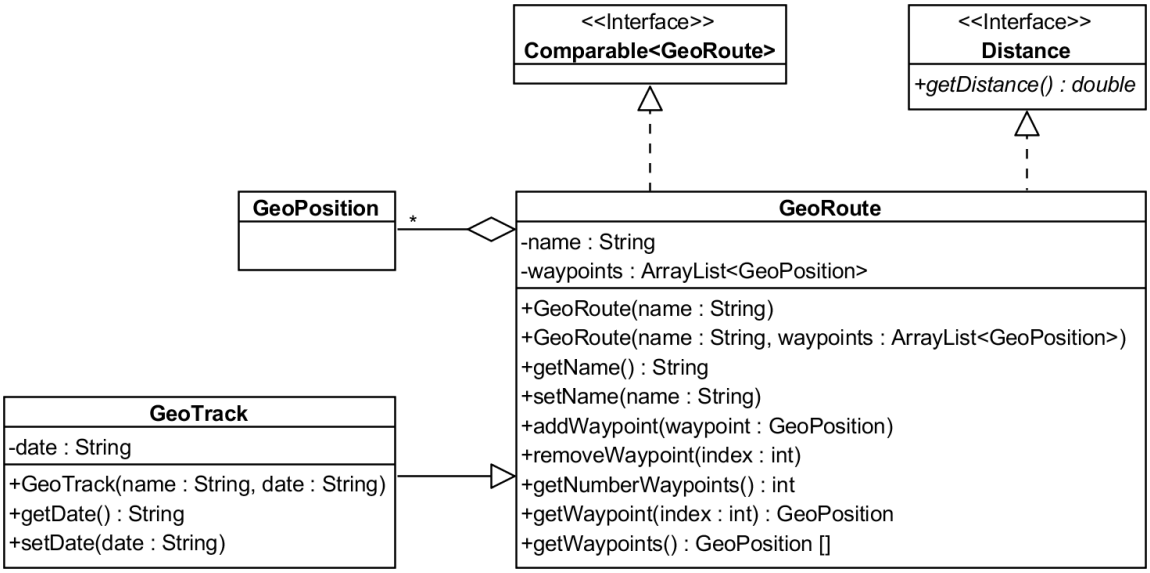
1 General Notes

The instructions given in lab exercise 1 apply. Please note also:

- Primary Learning Objectives: Lists of type *ArrayList*, Interfaces, Inheritance
- For the source code, please create a package called *lab2.geoPosition*.
- All instance variables have the modifier *private*.
- All given Unit-tests must be executed properly.
- Create constructors, getter, setter and the method *toString()* by using the menu *„Source/Generate …“*. If needed, adapt the methods tot he requirements.

2 Geographic Distances

You cannot always study, think about it yourself! How about a tour around the Alster or a flight? For this, let us create *routes* from geographic coordinates. The following UML diagram gives an overview of the classes to be created:



- A solid arrow points to the superclass, while a dashed arrow with a filled arrowhead points to an implemented interface.
- The empty diamond means that objects of class *GeoRoute* can contain objects of class *GeoPosition*. The number of contained objects is arbitrary by the specification *.

The classes and interfaces do have the following purpose:

- *GeoPosition* represents a location. Use the class created in lab exercise 1.
- *GeoRoute* represents a distance or a route between geographic locations.
- *GeoTrack* represents a (e.g. while jogging) recorded route.
- *RouteData* creates some routes to save typing work.
- The interface *Distance* allows to query a length.
- The interface *Comparable* is known from the lecture and provided by Java.

3 Functionality

3.1 Interface *Distance*

Requirement 1 The interface declares an abstract method *getDistance()* to return a distance.

Declaration:

- `double getDistance()`

3.2 Class *GeoRoute*

Requirement 2 A route has a name and an organized list of way points.

Requirement 3 The class has two constructors, one to which the name is passed and another one to which the name as well as a list of way points is passed.

Requirement 4 The name can be received by using a getter and be assigned a new value by using a setter.

Requirement 5 The method *addWaypoint()* adds a way point to the end of the list.

Requirement 6 The method *removeWaypoint()* removes a way point with index *i* from the list.

Requirement 7 The method *getNumberWaypoints()* returns the number of way point being in the list.

Requirement 8 The method *getWaypoint()* returns a reference to a way point with index *i*. Please remember that the first element of the list has the index 0.

Requirement 9 The method *getWaypoints()* returns an array with all way points contained in the list. (Please note: use one of the *toArray()*-methods of class *ArrayList*. How are the methods different? Which one should you use best?)

Requirement 10 The class implements the interface *Distance*. The method *getDistance()* returns the total distance of a route in kilometers.

Requirement 11 The class implements the interface *Comparable<GeoRoute>*. The criterion for the comparison is the total distance.

Requirement 12 The class overrides the method *toString()*. The return value has the format „`<Name> (<Gesamtstrecke> km)`“, whereas expressions in angle brackets are to be replaced by the corresponding values. The value of the total distance is output with one decimal place.

Declarations:

- `public GeoRoute(String name)`
- `public GeoRoute(String name, ArrayList<GeoPosition> waypoints)`
- `public String getName()`
- `public void setName(String name)`
- `public void addWaypoint(GeoPosition waypoint)`
- `public void removeWaypoint(int index)`
- `public int getNumberWaypoints()`
- `public GeoPosition getWaypoint(int index)`
- `public GeoPosition[] getWaypoints()`
- `public double getDistance()`
- `public int compareTo(GeoRoute other)`
- `public String toString()`

3.3 Class *GeoTrack*

Requirement 13 — The class extends the class *GeoRoute* by the date on which the represented route was covered. The date is stored as string having the format *yyyy-mm-dd* with year *y*, month *m* and day *d*.

Requirement 14 — Exactly one constructor is existing to which the name of the route and the date are passed.

Requirement 15 — The date can be received by using a getter and be assigned a new value by using a setter.

Declarations:

- `GeoTrack(String name, String date)`
- `public String getDate()`
- `public void setDate(String date)`

3.4 Unit-Tests

Requirement 16 — Ensure that all given Unit-tests are executed properly.

4 HAW-Running Group

You want to start a HAW running group. The following three routes are available: from the HAW around the Binnenalster or the Außenalster and back or around the city park.

Create an executable class to estimate the lengths of these routes. The class *RouteData* contains methods that generate routes around the Binnenalster or Außenalster. You still must create the route around the city park. Coordinates of the waypoints can be obtained, for example, by clicking on the corresponding positions in *Google Maps*. Choose as the starting point the connection of the Ohlsdorfer Straße to the Jahnkampfbahn and add some coordinates, which lead you in the largest possible round around the city park and back to the starting point. Transfer the length of the routes to the table below.

Route	*Length km*
Binnenalster	
Außenalster	
City Park	

5 Flight Routes

The method *createFlightRoutes()* of the class *RouteData* creates a list of different flight routes. Create an executable class that sorts the list in ascending order of the length of the flight routes and then outputs the routes and route names. (Please note: A list of type *ArrayList* list can be sorted using the class method *Collections.sort()*.)

6 Compliance with Coding Guidelines (Quality)

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

7 Theory Questions

1. Explain why the class *GeoRoute* uses a list instead of an array to reference the waypoints.
2. Could *GeoRoute* implement the method *getDistance()* without implementing the interface *Distance*?
3. Could *GeoRoute* implement the interface *Distance* without implementing the method *getDistance()*?
4. Which methods does the class *GeoTrack* have?
5. How can an object of class *GeoTrack* access the stored waypoints?