Objektorientierte Programmierung (E-B3-OPP)

Praktikumsaufgabe 2:
Geografische Wegstrecken (Routen)

HAW Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

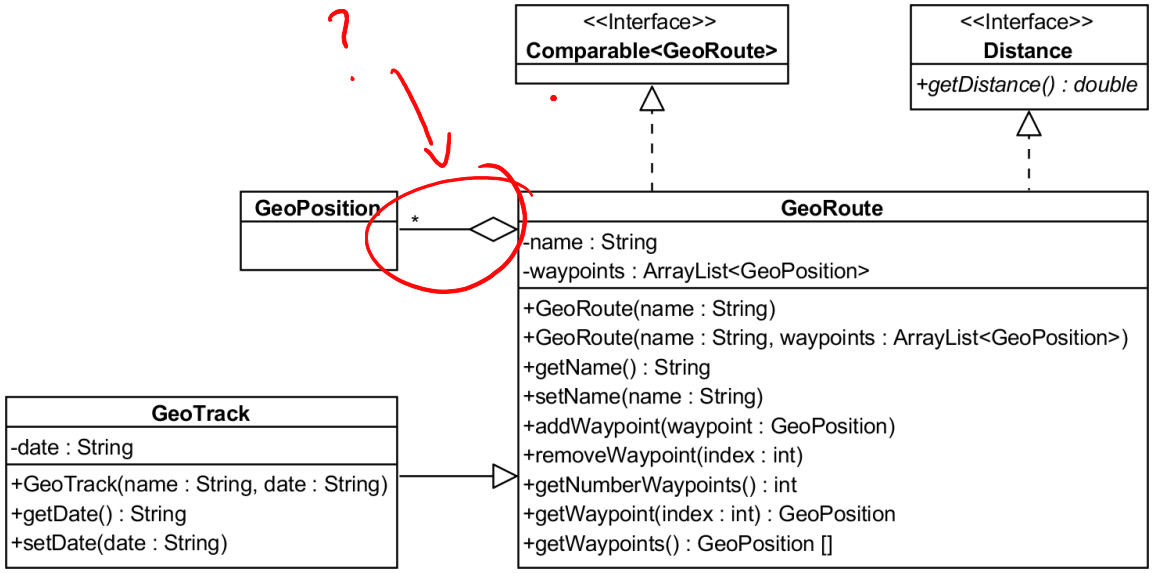
1 Allgemeine Hinweise

Es gelten die in Praktikumsaufgabe 1 angegebenen Hinweise. Beachten Sie zudem:

- Primäre Lernziele: Listen vom Typ *ArrayList*, Interfaces, Vererbung
- Legen Sie für die Quelltexte ein Paket namens *lab2.geoPosition* an.
- Alle Instanzvariablen besitzen den Modifier *private*.
- Die mitgelieferten JUnit-Tests müssen fehlerfrei durchlaufen.
- Erzeugen Sie Konstruktoren, Getter, Setter sowie die Methode *toString()* durch die Menüpunkte *„Source/Generate …“*. Passen Sie die Methoden gegebenenfalls den Anforderungen an.

2 Geografische Wegstrecken

Man kann nicht immer nur studieren, denken Sie auch mal an sich! Wie wäre es mit einer Runde um die Alster oder einer Flugreise? Lassen Sie uns hierzu aus geografischen Koordinaten Wegstrecken bzw. *Routen* erzeugen. Nachfolgendes UML-Diagramm gibt einen Überblick der zu erstellenden Klassen:



- Ein durchgezogener Pfeil zeigt zur Basisklasse, während ein gestrichelter Pfeil mit ausgefüllter Pfeilspitze zu einem implementierten Interface zeigt.
- Die leere Raute bedeutet, dass Objekte der Klasse *GeoRoute* Objekte der Klasse *GeoPosition* enthalten können. Die Anzahl der jeweils enthaltenen Objekte ist durch die Angabe * beliebig.

Folgenden Zweck haben die Klassen und Interfaces:

- *GeoPosition* repräsentiert einen Ort. Verwenden Sie die in Praktikum 1 erstellte Klasse.
- *GeoRoute* repräsentiert eine Wegstrecke bzw. Route aus geografischen Orten.
- *GeoTrack* repräsentiert eine (z.B. beim Joggen) aufgezeichnete Wegstrecke.
- *RouteData* erzeugt einige Wegstrecken, um Ihnen Tipparbeit zu ersparen.
- Das Interface *Distance* erlaubt die Abfrage einer Länge.
- Das Interface *Comparable* ist aus der Vorlesung bekannt und durch Java vorgegeben.

3 Funktionalität

3.1 Interface *Distance*

Requirement 1 — Die Schnittstelle deklariert eine abstrakte Methode *getDistance()* zur Rückgabe einer Entfernung.

Deklarationen:

- `double getDistance()`

3.2 Klasse *GeoRoute*

Requirement 2 — Eine Route hat einen Namen sowie eine geordnete Liste von Wegpunkten.

Requirement 3 — Die Klasse besitzt je einen Konstruktor, dem der Name übergeben wird, sowie einen Konstruktor, dem sowohl der Name als auch eine Liste von Wegpunkten übergeben wird.

Requirement 4 — Der Name kann über einen Getter erfragt sowie über einen Setter zugewiesen werden.

Requirement 5 — Die Methode *addWaypoint()* fügt einen Wegpunkt am Ende der Liste hinzu.

Requirement 6 — Die Methode *removeWaypoint()* entfernt den Wegpunkt mit Index *i* aus der Liste.

Requirement 7 — Die Methode *getNumberWaypoints()* gibt die Anzahl der Wegpunkte in der Liste zurück.

Requirement 8 — Die Methode *getWaypoint()* gibt eine Referenz auf den Wegpunkt mit Index *i* zurück. Beachten Sie, dass das erste Element der Liste den Index 0 hat.

Requirement 9 — Die Methode *getWaypoints()* gibt ein Array aller in der Liste enthaltener Wegpunkte zurück. (Hinweis: Verwenden Sie eine der *toArray()*-Methoden der Klasse *ArrayList*. Wie unterscheiden sich die Methoden? Welche sollten Sie am besten verwenden?)

Requirement 10 — Die Klasse implementiert das Interface *Distance*. Die Methode *getDistance()* gibt die Gesamtstrecke der Route in Kilometern zurück.

Requirement 11 — Die Klasse implementiert das Interface *Comparable<GeoRoute>*. Das Kriterium für den Vergleich ist die Gesamtstrecke.

Requirement 12 — Die Klasse überschreibt die Methode *toString()*. Die Rückgabe hat die Formatierung „`<Name> (<Gesamtstrecke> km)`“, wobei Ausdrücke

in spitzen Klammern durch die entsprechenden Werte zu ersetzen sind. Die Gesamtstrecke ist auf eine Stelle hinter dem Komma auszugeben.

Deklarationen:

- `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 Klasse *GeoTrack*

Requirement 13 — Die Klasse erweitert die Klasse *GeoRoute* um das Datum, an dem die repräsentierte Strecke zurückgelegt wurde. Das Datum wird als Zeichenkette im Format *yyyy-mm-dd* mit Jahr *y*, Monat *m* und Tag *d* gespeichert.

Requirement 14 — Es existiert genau ein Konstruktor. Diesem werden ein Name der Strecke sowie das Datum übergeben.

Requirement 15 — Das Datum kann über einen Getter erfragt sowie über einen Setter zugewiesen werden.

Deklarationen:

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

3.4 Unit-Tests

Requirement 16 — Stellen Sie sicher, dass alle gegebenen Unit-Tests fehlerfrei laufen.

4 HAW-Lauftreff

Sie wollen einen HAW-Lauftreff ins Leben rufen. Zur Auswahl stehen folgende drei Laufstrecken: von der HAW um die Binnenalster oder die Außenalster und zurück oder aber um den Stadtpark.

Erstellen Sie eine ausführbare Klasse, um die Längen dieser Strecken abzuschätzen. Die Klasse *RouteData* enthält hierfür Methoden, die Strecken um die Binnen- bzw. Außenalster erzeugen. Die Route um den Stadtpark müssen Sie hingegen noch erstellen. Koordinaten der Wegpunkte erhalten Sie beispielsweise, indem Sie in *Google Maps* auf die entsprechenden Positionen klicken. Wählen Sie als Ausgangsort die Verbindung der Ohlsdorfer Straße zur Jahnkampfbahn und fügen Sie einige Koordinaten hinzu, die Sie in einer möglichst großen Runde durch den Stadtpark zurück zum Ausgangspunkt führen. Übertragen Sie die Länge der Routen in die nachfolgende Tabelle.

Route	*Länge km*
Binnenalster	
Außenalster	
Stadtpark	

5 Flugrouten

Die Methode *createFlightRoutes()* der Klasse *RouteData* erzeugt eine Liste mit unterschiedlichen Flugrouten. Erstellen Sie eine ausführbare Klasse, die die Liste nach aufsteigender Strecke der Flugroute sortiert und anschließend die Flugrouten mit Namen und Strecke ausgibt. (Hinweis: Eine Liste vom Typ *ArrayList* lässt sich über die Klassenmethode *Collections.sort()* sortieren.)

6 Einhaltung der Programmierrichtlinien (Qualität)

Stellen Sie sicher, dass alle in der über EMIL zur Verfügung gestellte Checkliste zur Softwarequalität aufgeführten Qualitätskriterien erfüllt sind.

7 Theoretische Fragen

1. Erläutern Sie, warum die Klasse *GeoRoute* zum Referenzieren der Wegpunkte kein Feld, sondern eine Liste verwendet.
2. Könnte *GeoRoute* die Methode *getDistance()* implementieren, ohne das Interface *Distance* zu implementieren?
3. Könnte *GeoRoute* das Interface *Distance* implementieren, ohne die Methode *getDistance()* zu implementieren?
4. Welche Methoden besitzt die Klasse *GeoTrack*?
5. Wie kann ein Objekt der Klasse *GeoTrack* auf die gespeicherten Wegpunkte zugreifen?