Implementierung

Praxis der Softwareentwicklung 13
Entwicklung eines Fahrradroutenplaners
Team 16

Sven Esser, Manuel Fink, Thomas Keh,
Tilman Väth, Lukas Vojković, Fabian Winnen

WS 2011/2012

Inhaltsverzeichnis

1 Einleitung

Während der Implementierung von BiKeIT mussten einige Veränderungen am zuvor fertiggestellten Entwurf vorgenommen werden. Dies betrifft zum einen Dinge, die mangels Relevanz noch nicht ausmodelliert waren, und zum anderen Methoden und Klassen, bei denen sich erst während der Implementierung gezeigt hat, dass diese benötigt werden. An manchen Stellen wurde auch um des guten Programmierstiles willen eine Änderung vorgenommen.

Im folgenden werden diese Änderungen und Erweiterungen im Detail beschrieben. Des Weiteren wurden schon jetzt einige Komponententests durchgeführt, die mitsamt Abdeckungsstatistik am Ende des Dokumentes zu finden sind.

2 Änderungen am Entwurf

2.1 Änderungen am Entwurf des MapModels und der Datentypen

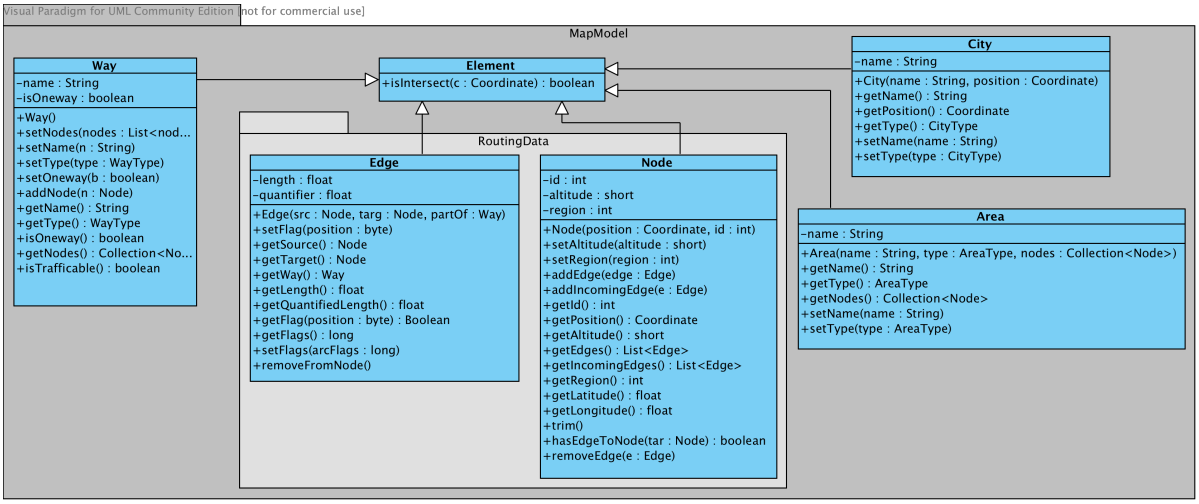
Um eine performante und strukturell einheitliche Implementierung der Elemente des Kartenmaterials zu gewährleisten, mussten Klassen in den Paketen MapModel, DataTypes sowie Utilites teilweise abgeändert werden. Außerdem mussten aus dem selben Grund neue Klassen eingeführt werden.

2.1.1 Klasse ArcFlags entfernt

Die Klasse ArcFlags wurde entfernt, um Arbeitsspeicher zu sparen. Die Arc-Flags[2] einer Kante werden nun direkt als Zahlenwert gespeichert, statt als ArcFlags-Objekt. Es wurde damit eine Einsparung von 20% erreicht.

Paket: MapModel
Betroffene Klasse: Edge

2.1.2 Neue Klasse Element



Die abstrakte Klasse Element repräsentiert ein Kartenelement, dh. einen Weg, eine Kante, einen Knoten, eine Stadt oder ein Gebiet. Sie bietet hiermit eine einheitliche

Schnittstelle, mit der überprüft werden kann, ob sich das gegebene Element in einem rechteckigen Koordinatenbereich befindet.

Paket: MapModel
Vererbt an: Area, City, Edge, Node, Way

2.1.3 Klasse Area

Der Klasse Area wurden neue Set-Methoden für Name und Typ hinzugefügt, da diese Informationen nicht immer an der gleichen Stelle in der OSM-Datei[13] liegen und somit gegebenenfalls nachträglich gesetzt werden müssen. Außerdem wurde die Methode isTraversable() entfernt. Plätze werden zwecks Performanz nun schon zur Vorberechnungszeit in Wege zerlegt.

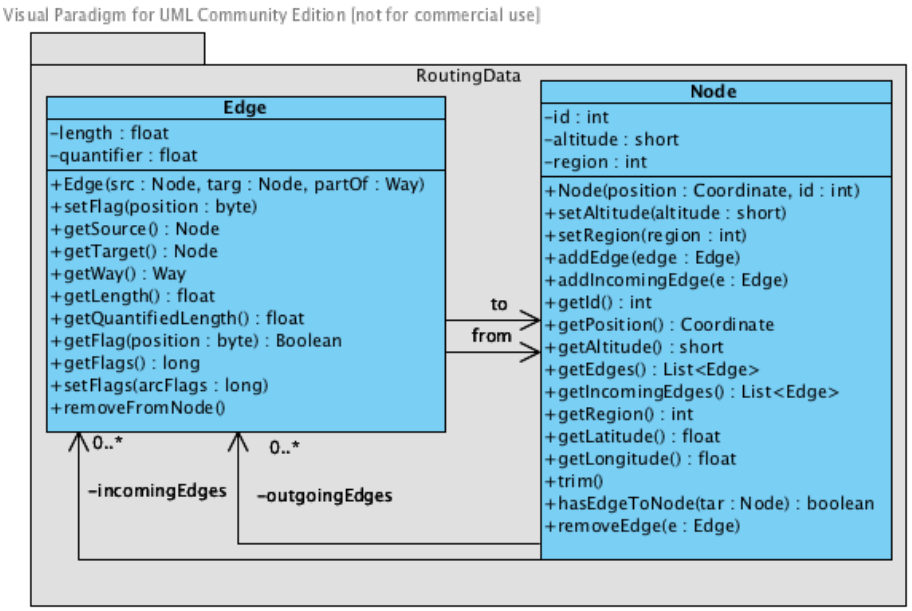
Paket: MapModel
Betroffene Klasse: OSMHandler

2.1.4 Klasse Way

Die Methode setNodes() wurde hinzugefügt, um in einem Schritt alle betreffenden Knoten zu übergeben. Die Klasse besitzt nun außerdem die Methode isTrafficable(), um bei der Routenberechnung mit dem Fahrrad befahrbare von nicht befahrbaren Straßen zu unterscheiden.

Paket: MapModel
Betroffene Klassen: Dijkstra, PreprocessingDijkstra, Importer

2.1.5 Änderungen am Paket RoutingData



Klasse Edge

Die Klasse Edge gibt mit getQuantifiedLength() die gewichtete Länge zurück, womit die Methode getQuantifier nur noch privat genutzt wird. Die Methode setQuantifier wurde entfernt, da die Gewichtungen anhand des Wegtyps dynamisch ermittelt werden.
Zum schnelleren Speichern und Laden der Arc-Flags[2] einer Kante gibt es die Methoden setFlags und getFlags welche alle Flags auf einmal setzen bzw. auslesen.

Paket: MapModel
Betroffene Klassen: Dijkstra, PreprocessingDijkstra, Importer

Klasse Node

Die Klasse Node hat eine entscheidende Änderung erfahren. Im Anwendungsprogramm enthält sie nur Referenzen[12] auf ausgehende Kanten, wie sie von der Klasse Dijkstra benötigt werden und im Vorberechnungsprogramm enthält sie nur noch eingehende Kanten, wie sie die Arc-Flags-Berechnung mit der Klasse PreprocessingDijkstra benötigt. So konnten sowohl eine erhebliche Menge Arbeitsspeicher pro Knoten gespart werden, als auch zusätzlicher Aufwand für den Rückwärts-Dijkstra[15] vermieden werden. Die Schnittstelle trägt dem mit den zusätzlichen Methoden setIncomingEdge() und getIncomingEdges() Rechnung.
Der Längen- und Breitengrad kann nun jeweils einzeln abgefragt werden.
Mit hasEdgeToNode(target) kann überprüft werden, ob der Knoten zu einem anderen adjazent ist. Dies wird für die Regionenaufteilung mit METIS[11] benötigt, um die von

diesem Programm geforderten Rückkanten zu erzeugen, falls die betrachtete Kante zu einer Einbahnstraße gehört.
Die Methode removeEdge() wird von einer Kante, welche sich selbst vom Knoten löschen möchte, aufgerufen
Mit trim() wird die Liste der Kanten auf genau die Größe der enthaltenen Referenzen[12] gekürzt, so dass weniger Speicher verbraucht wird.
Die Höhendaten werden, da der höchste Punkt der Erde nur 8848 Meter hoch ist in dem kleineren Ganzzahlformat short[16] gespeichert.

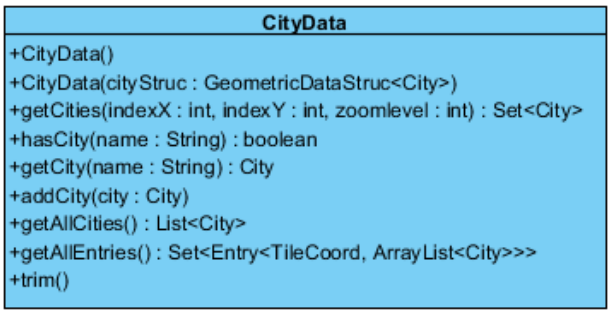
Paket: MapModel
Betroffene Klassen: PreprocessingDijkstra, Exporter, Importer, AltitudeMapController

2.1.6 Klasse City

Da es passieren kann, dass in dem OSM-Datensatz[13] der Name erst ermittelt wird, nachdem das dazugehörende City-Objekt erzeugt wurde, erlaubt die Methode setName(), den Namen nachträglich zu setzen.
Die Methoden setType() und getType() ordnen der Stadt eine Größenordnung zu.

Paket: MapModel
Betroffene Klasse: OSMHandler

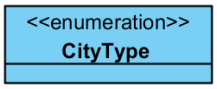
2.1.7 Neue Klasse CityData



Die Klasse CityData verwaltet die Städte in einer geometrischen Struktur, um beim Zeichnen performant an alle zu zeichnenden Städtenamen für eine Kachel zu gelangen.

Paket: MapModel
Enthält: City
Benötigt von: MapModel

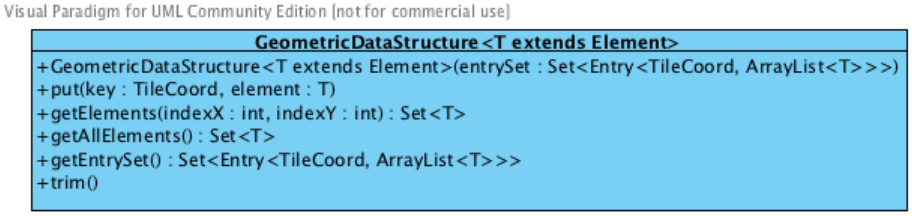
2.1.8 Neue Enumeration CityType



Die Enumeration CityType beschreibt den Typ einer Stadt - wie z.B. Dorf. Es gibt dabei eine vordefinierte Menge an möglichen Stadttypen. Dies wird benötigt um die Darstellung der Städtenamen ihrer Größenordnung anzupassen.

Paket: DataTypes
Benötigt von: City

2.1.9 Umbenennung der Klasse 2DDataStructure



Die Klasse 2DDataStructure wurde umbenannt in „GeometricDataStructure“, da Java[9] Klassennamen mit einer Ziffer als erstes Zeichen nicht akzeptiert.

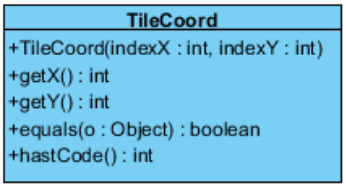
Paket: Utilities
Betroffene Klassen: Importer, CityData, NodeData, StreetData, TerrainData

2.1.10 Umbenennung der Klasse Point

Die Klasse Point wurde umbenannt in Pixel, da der Name besser ausdrückt, dass die Klasse ein bestimmtes Pixel auf der Karte repräsentiert.

Paket: DataTypes
Betroffene Klassen: CalculatedRoute, MercadorProjection

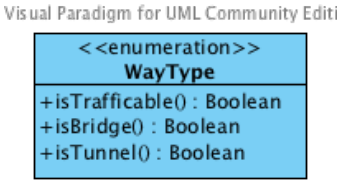
2.1.11 Klasse TileCoord



Die Klasse TileCoord repräsentiert den Index einer Kachel, der unter Anderem für die geeignete Ablage in der geometrischen Datenstruktur vonnöten ist.

Paket: DataTypes
Betroffene Klassen: Importer, MapTileRenderer, CityData, NodeData, StreetData, TerrainData, GeometricDataStructure, Exporter, MercadorProjection

2.1.12 Enumeration WayType



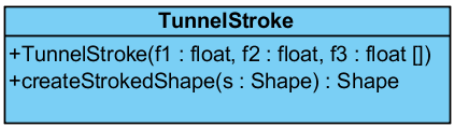
Die Enumeration WayType wurde um die Methoden isTrafficable(), isTunnel() und isBridge() erweitert. Ersteres wird von der Klasse Dijkstra benötigt, um befahrbare von unbefahrbaren Straßen zu unterscheiden und die beiden letzten Methoden ermöglichen einen angepassten Renderingstil, wenn ein Weg sich auf einer Brücke bzw. in einem Tunnel befindet.

Paket: DataTypes
Betroffene Klassen: MapTileRenderer, Way

2.2 Änderungen am Entwurf des Renderingprozesses

Da sich der Renderingprozess[14] als sehr komplex herausgestellt hat, mussten einige Erweiterungen am bisherigen Entwurf vorgenommen werden. So konnte der Code auf inhaltlich sinnvolle Klassen verteilt werden, was auch eine höhere Robustheit des Codes verspricht.

2.2.1 Neue Klasse TunnelStroke

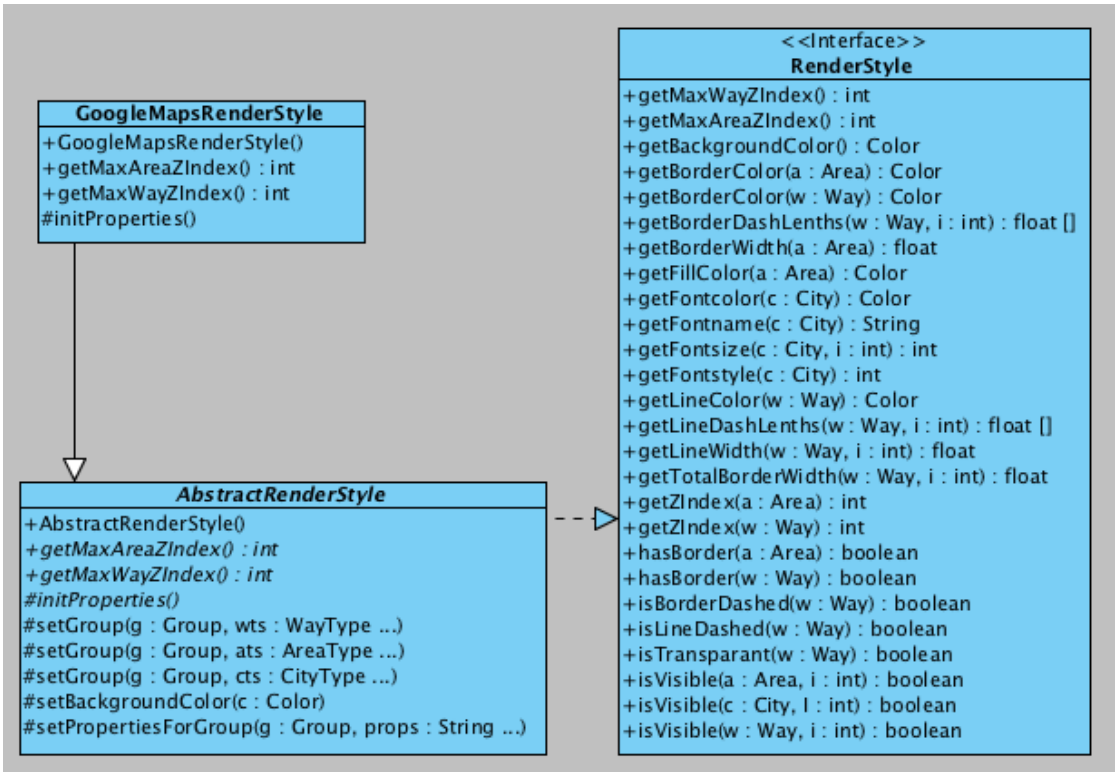


Die Klasse TunnelStroke beschreibt, wie ein Tunnel gezeichnet wird.

Paket: Utilities
Benötigt von: MapTileRenderer

2.2.2 RenderStyle

Um das Erscheinungsbild von Elementen auf der Karte zu beeinflussen und besser zu verwalten, wurden drei neue Klassen dem Paket Utilities hinzugefügt.



Neue Schnittstelle RenderStyle

Die Schnittstelle RenderStyle bietet Methoden, um Eigenschaften der Darstellung von Objekten der Klassen Way, Area und City abzufragen.

Paket: Utilities
Benötigt von: MapTileRenderer

Neue Klasse AbstractRenderStyle

Die abstrakte Klasse AbstractRenderStyle implementiert die Schnittstelle RenderStye. Intern werden die anzeigbaren Objekte (Way, Area, City) in Gruppen verwaltet. Jeder Gruppe werden in Form von Strings[18] Eigenschaften zugeordnet, welche über die implementierte Schnittstelle in passender Form interpretiert und abgefragt werden können. Die Klasse AbstractRenderStyle bietet Methoden, um diese Gruppen zu erstellen und ihnen die Eigenschaften zuzuordnen.

Paket: Utilities
Benötigt von: MapTileRenderer
Implementiert: RenderStyle

Neue Klasse GoogleMapsRenderStyle

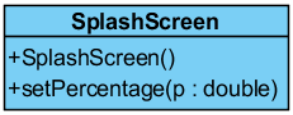
Benötigt von: MapRendering Die Klasse GoogleMapsRenderStyle ist eine Spezialisierung von AbstractRenderStyle und repräsentiert ein konkretes Erscheinungsbild, indem es in den überschriebenen Methoden eigene Gruppen erstellt und ihnen konkrete Eigenschaften zuordnet. Wie der Name der Klasse ahnen lässt, orientiert sich das Erscheinungsbild an Google Maps[6].

Paket: Utilities
Benötigt von: MapTileRenderer
Erbt von: AbstractRenderStyle

2.3 Änderungen am Entwurf der Benutzerschnittstelle

Die Benutzerschnittstelle war noch nicht vollkommen ausmodelliert. Die fehlenden Klassen werden hier beschrieben. Außerdem wurden Anpassungen zur Vermeidung von Coderedundanz[3] vorgenommen.

2.3.1 Neue Klasse SplashScreen



Die Klasse SplashScreen stellt einen Ladebalken bereit, der beim Programmstart angezeigt wird. Die Klasse Importer aktualisiert hierbei in geeigneten Abständen den aktuellen Ladestand.

Paket: UserInterface
Benötigt von: Importer

2.3.2 Neue Klasse AboutDialog

Diese Klasse öffnet ein Fenster, das das Projektlogo und die Namen der Entwickler anzeigt. Es wird von der MainGUI erzeugt.

Paket: UserInterface
Benötigt von: MainGUI

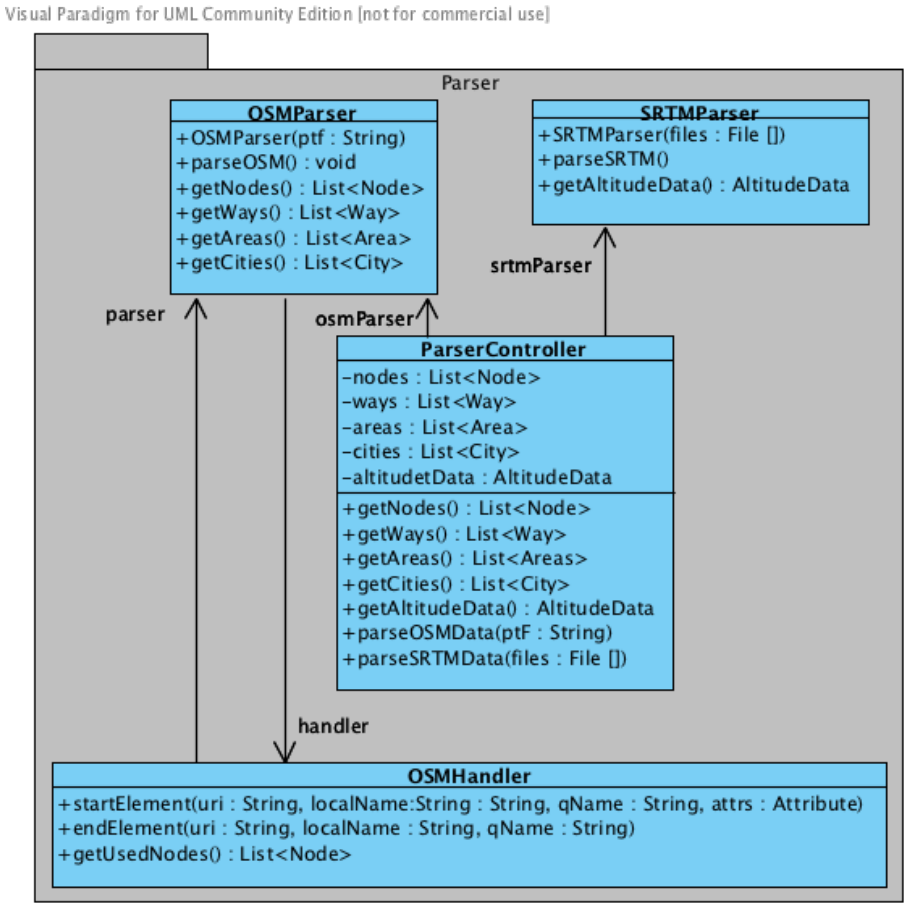
2.3.3 Neue Klasse GUIUtilities

Die Klasse GUIUtilities bietet Schnittstellen zur Formatierung von Längen- und Zeitangaben, welche an verschiedenen Stellen zur Ausgabe von Informationen zu einer berechneten Route benötigt werden. Die Änderung würde nötig, um doppelten Code zu vermeiden.

Paket: Utilities
Benötigt von: MainGUI, DescriptionGUI, PrintingController

2.4 Änderungen am Entwurf des Parsers

Auch der Parser erfuhr einige Änderungen. Zum größten Teil rühren diese von der Anbindung der Java-Bibliothek SAXParser, die die Impelementierung von einigen Schnittstellen vorschreibt. Außerdem wurden die Methoden an die verwendete geometrische Datenstruktur angepasst.



2.4.1 Neue Klasse OSMHandler

Der Java-eigene SAXParser zum Einlesen von XML-Daten[19], benötigt einen Handler, der die Weiterverarbeitung der sequenziell ankommenden Daten bestimmt. Der OSM-Handler enthält Methoden, die bei jedem öffnenden und schließenden XML-Element[19] aufgerufen werden. In diesen werden alle relevanten Wege und Knoten gefiltert und daraus Objekte für Wege, Gebiete, Städte und Knoten erzeugt.

Paket: Parser

Benötigt von: OSMParser, SAXParser (Java-intern)
Erbt von: DefaultHandler (Java-intern)

2.4.2 Klasse OSMParser

Dem Konstruktor wird nun ein String[18] übergeben, welcher den Pfad zur einzulesenden OSM-Datei enthält und direkt vom SAXParser verarbeiteten werden kann.

2.4.3 Klasse SRTMParser

Durch eine Änderung des Konstruktors, der nun ein Array aus Dateien entgegennimmt, wird die Bearbeitung von mehreren SRTM-Dateien[17] ermöglicht, was bei größeren Karten nötig wird.

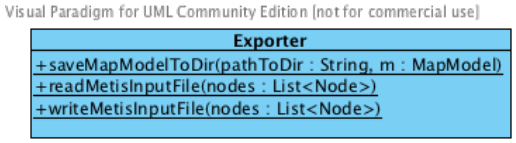
2.4.4 Klasse ParserController

Die Methoden parseOSMData() und parseSRTMData() wurden entsprechend der Änderungen an den Klassen OSMParser und SRTMParser angepasst.

2.5 Sonstige Entwurfsänderungen

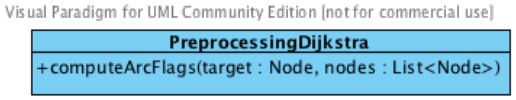
Es gab noch einige weitere Änderungen und Erweiterungen am bisherigen Entwurf. Dies betrifft unter Anderem Utility-Klassen die sich als nützlich erweisen haben. Außerdem wurde nun die Druckfunktion ausmodelliert, was zuvor ausgespart wurde.

2.5.1 Klasse Exporter



Für die Benutzung von METIS[11] wurden die Methoden readMetisInputFile() und writeMetisInputFile() hinzugefügt.

2.5.2 Klasse PreprocessingDijkstra



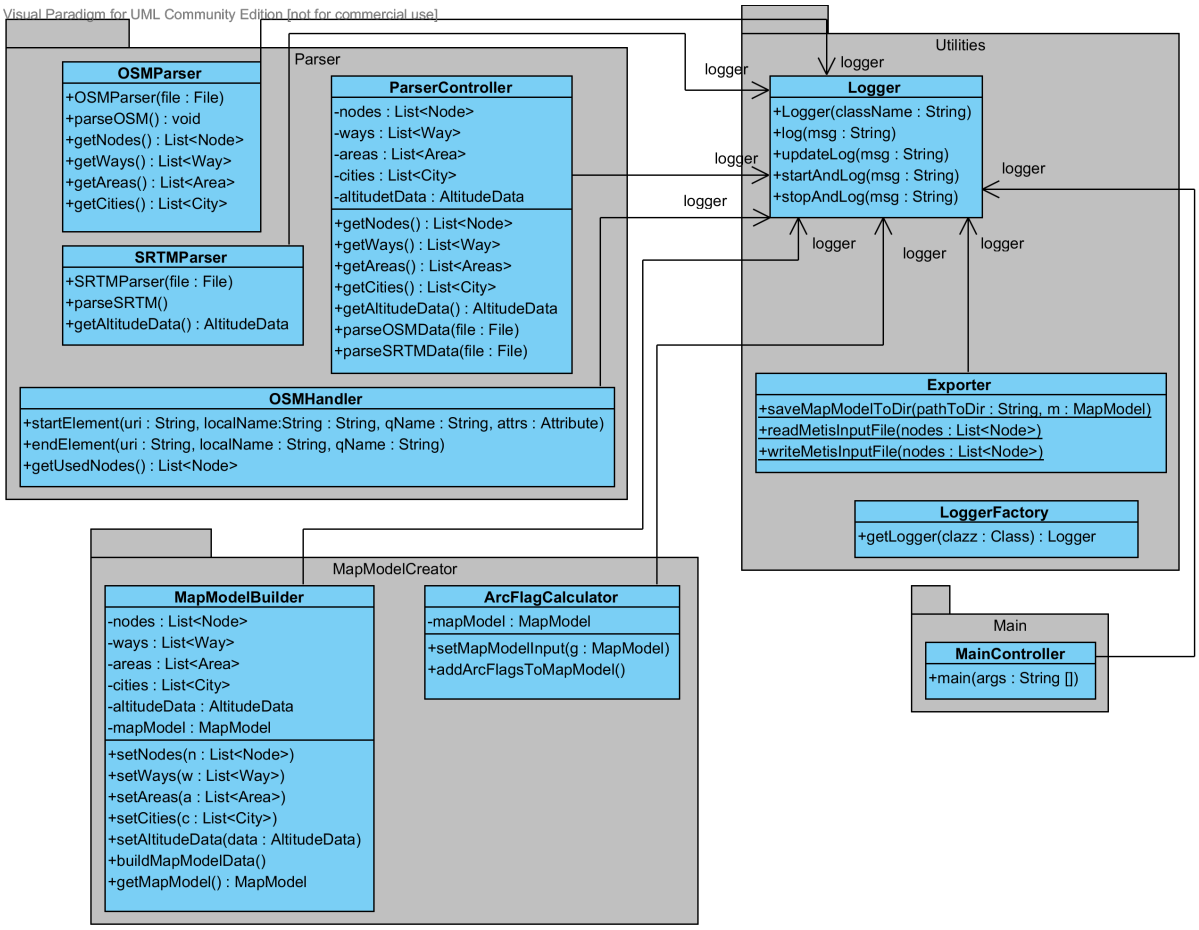
Die Methode computeArcFlags() erwartet nicht mehr das gesammte MapModel sondern eine Liste von Knoten.

2.5.3 Umbenennung der Klasse Initializer

Die Klasse Initializer wurde umbenannt in „Importer“, um die semantische Verwandtschaft zu der Klasse Exporter zu verdeutlichen. Außerdem wurde sie ins Paket Main verschoben, um Abhängigkeiten zwischen Vorberechnungsprogramm und Anwendungsprogramm zu vermeiden.

Pakete: Utilities, Main
Betroffene Klasse: MainGUI

2.5.4 Neue Klasse Logger



Die Klasse Logger wurde neu eingeführt, um die Ausgaben auf der Kommandozeile der verschiedenen Klassen des Vorberechnungsprogramms zu vereinheitlichen und übersichtlich darzustellen. Sie ermöglicht präzise Zeitangaben und die Darstellung des Berechnungsfortschritts.

Paket: Utilities
Benötigt von: MainController, MapModelBuilder, OSMParser, ParserController, SRTMParser

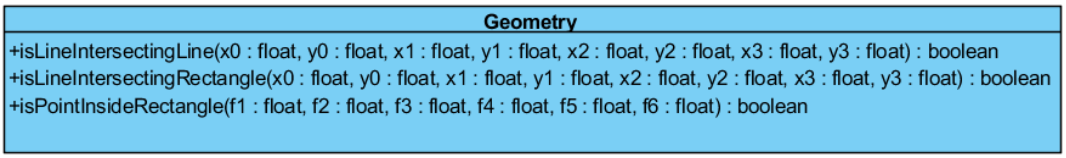
2.5.5 Neue Klasse LoggerFactory

Die LoggerFactory-Klasse erzeugt für unterschiedliche Klassen jeweils eine eigene Logger-Instanz. Diese Vorgehensweise ist nötig, damit auf der Kommandozeile auch die Klasse, aus der der Aufruf kam, ausgegeben werden kann.

Paket: Utilities

Benötigt von: MainController, MapModelBuilder, OSMParser, ParserController, SRTMParser

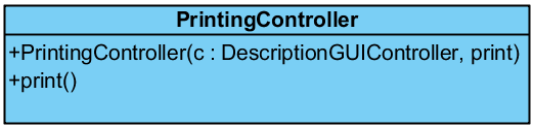
2.5.6 Neue Klasse Geometry



Die Klasse Geometry dient zur Durchführung geometrischer Berechnungen. Alle Methoden sind statisch, so dass diese von jeder Klasse aufgerufen werden können, die diese Methoden in Anspruch nehmen müssen. Hauptsächlich dienen diese dazu, herauszufinden, welche Kacheln die Kartenelemente schneiden.

Paket: DataTypes
Benötigt von: Way, Edge, Area

2.5.7 Neue Klasse PrintingController



Der PrintingController ermöglicht das Drucken der Routenbeschreibung. Zum Erzeugen des Dokuments nutzt er die gleichen Zugriffsmethoden wie die DescriptionGUI und erbt hierzu vom DescriptionGUIController.

Paket: UserInterface
Erbt von: DescriptionGUIController
Benötigt von: DescriptionGUIController

3 Komponententests

Schon während der Implementierungsphase wurden automatisierte Komponententests[10] entwickelt und durchgeführt. Ziel war, relevante Programmteile zu verifizieren und dabei eine hohe Codeüberdeckung[4] zu erreichen.

3.1 Test der RenderingStyle-Komponente

Beteiligte Klassen: RenderingStyle, AbstractRenderingStyle
Erreichte Codeüberdeckung: 76,2%

Das Modul RenderingStyle, welches mit einfachen Strings Rendering-Parameter für Elemente der Klassen *Way*, *Area* und *City* setzt, verwaltet und für sie eine Schnittstelle bereitstellt, wurde mithilfe einer Unterklasse von AbstractRenderingStyle getestet. In diesen Tests wurde vorrangig getestet, ob alle definierten Strings[18] erkannt, und korrekt interpretiert wurden.
Insgesamt wurden alle Methoden von den Tests überdeckt. Aufgrund des Umfangs der Klasse, und der vielen Variationen und Fehlerüberprüfungen haben wir uns deshalb mit einer Codeüberdeckung[3] von 76,2% zufrieden gegeben.

3.2 Test der Import-Export-Komponente

Beteiligte Klassen: Importer, Exporter, alle Klassen des MapModel-Pakets
Erreichte Codeüberdeckung Importer: 93,3%
Erreichte Codeüberdeckung Exporter: 78,4%
Erreichte Codeüberdeckung MapModel-Paket: 45,7%

Die Import-Export-Komponente wurde mithilfe einer erstellten Instanz von MapModel getestet. Das Test-MapModel wurde exportiert und anschließend neu aus den Binärdateien geladen. Danach wurde mittels der überschriebenen equals-Methode des MapModels überprüft, ob alle MapModel-Daten korrekt im-/exportiert wurden. Durch dieses Verfahren wurde zusätzlich eine durchschnittliche Codeüberdeckung[3] aller Klassen des MapModel-Pakets von 45,7% erreicht.

Das Test-MapModel wird automatisch aus 100 Knoten erzeugt, die als doppelter Ring verbunden werden. Aus diesen Knoten wird ein Way und ein Area erstellt und mit zusätzlichen Test-Daten ergänzt. Da auch Städte mit verschiedenen Typen hinzugefügt werden, ist das Test-MapModel ein repräsentatives Testobjekt.

3.3 Test der Geometry-Klasse

Beteiligte Klassen: Geometry
Erreichte Codeüberdeckung: 100%

Die Klasse Geometry, welche unabhängig von anderen Klassen geometrische Berechnungen statisch durchführt, wurde mit einfachen Beispiel-Daten automatisch getestet. Mit 16 Tests für die Methoden

- *isLineIntersectingLine(...)*,
- *isPointInsideRectangle(...)*,
- *isLineIntersectingRectangle(...)*

konnte eine Codeüberdeckung[3] von 100% erreicht werden.

3.4 Test der Dijkstra-Klasse

Beteiligte Klassen: Dijkstra, Node, Edge
Erreichte Codeüberdeckung: 85,7%

Die Klasse Dijkstra[5] wurde mithilfe eines, von Hand erstellten Graphen aus fünf Knote und sieben Kanten auf ihre Korrektheit getestet. Es wurde primär getestet ob die kürzeste Pfad gefunden wird. Dabei wurde eine Codeüberdeckung[3] von 85,7% erreicht. Dies liegt daran, dass es im Test-Graphen keine Arc-Flags[2] und keine unpassierbare Ways enthält.

4 Glossar

[1] **Altitude** Höhe über Normal-Null

[2] **Arc-Flags** Beschleunigungstechnik für den Dijkstra Algorithmus zur schnelleren Suche des kürzesten Pfades

[3] **Coderedundanz** mehrfaches Auftreten gleichen Codes

[4] **Codeüberdeckung** Code wird überdeckt wenn er im laufenden Programm aufgerufen wird

[5] **Dijkstra-Algorithmus** Algorithmus zur Berechnung einer kürzesten Route zwischen zwei Punkten, Name geht auf Erfinder Edsger W. Dijkstra zurück

[6] **GoogleMaps** Dienst des Unternehmens Google Inc., der es ermöglicht Orte,Objekte etc. zu suchen und auf einer Karte oder auf einem Bild der Erdoberfläche anzuzeigen

[7] **GUI** Benutzeroberfläche

[8] **int** 32-Bit Datentyp zum Speichern ganzzahliger Werte

[9] **Java** Programmiersprache

[10] **Komponententest** Test eines Teils des Programms. Dies kann auch nur den Test einer einzelnen Klasse beinhalten

[11] **METIS** Programm das eine Karte in Bereiche aufteilt

[12] **Referenz** Verweis auf ein Objekt

[13] **OSM** steht für OpenStreetMap. Dies ist ein für jeden zugängliches Projekt, das weltweite geographische Daten sammelt

[14] **Renderingprozes** Prozess des Zeichnens der Karte/Route

[15] **Rückwärts-Dijkstra** Ein Dijkstra auf einem Graphen mit umgekehrten Kanten

[16] **short** 16-Bit Datentyp zum Speichern ganzzahliger Werte

[17] **SRTM-Bilddatei** Digitale Karte die Informationen über Höhenmeter enthält

[18] **String** Zeichenkette

[19] **XML** steht für Extensible Markup Language. Dies ist eine ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdaten

[20] **Zoomen** Vergrößern/Verkleinern eines Bildschirmausschnittes

[21] **Zoomstufe** gibt den Grad der Vergrößerung/Verkleinerung an