Fahrtroutenplanung Projektdokumentation zum

Softwareentwicklungsprojekt

(Entwicklerdokumentation)

Lehrveranstaltung „Software Engineering I / II"

1. Juli 2016

Entwickler: Maxim Aisel, Paul Wolff, Dennis Hinterwimmer

Auftraggeber: Prof. Dr.-Ing. habil. Hartmut Fritzsche

Bachelorstudiengang Medieninformatik
Hochschule für Technik und Wirtschaft Dresden

Zusammenfassung

Beim der hier beschriebenen Software handelt es sich um einen Fahrtroutenplaner. Dieser muss in der Lage sein, ausgehend von einem vom Nutzer gewählten Lager, eine Route zu finden, die alle Lager miteinander verbindet. Die Route soll ähnlich einer Liste visualisiert werden und die einzelnen Lager nachfolgend anzeigen. Zusätzlich soll die Wegverbindung in Kilometern angezeigt werden. Die Daten liegen in Form einer Menge von Lagern, mit Verbindungen untereinander, vor.

Inhaltsverzeichnis

1 Einleitung

In einer globalisierten, dezentralisierten Produktion besteht an vielen Stellen ein hoher Optimierungsbedarf. Insbesondere die Wegeplanung wächst schnell zu einem komplexen Problem an. Der Fahrtroutenplaner soll dazu dienen ab einem bestimmten Lager eine Strecke zu ermitteln, welche durch alle weiteren Lager führt und nach Möglichkeit die kürzeste Strecke ermittelt. Für die Entwicklung war es notwendig, die Hauptprobleme zu identifizieren, zu klassifierieren, zu analysieren und in Teilprobleme zu zerlegen. Die dabei entstandene Software kann über die Wegplanung entfernter Lager hinaus, für viele verwandte Probleme genutzt werden, beispielsweise für das Straßenbahnnetz.

2 Projektmanagement

2.1 Vorgegebener Zeitablauf

Vorlage Pflichtenheft	01.05.2016
Bearbeitung Analyse/Entwurf:	
Klassendiagramme, Komponentendiagramme, Paketstruktur; Verteilungsdiagramme, ... Festlegung der Rollenverteilung	

Vorlage Projektdokumentation (Zwischenstand)	18.05.2016
Durchführung der Implementierung User Interface Prototyping	

Gruppenkolloquium	21.05.2016
Implementation, Testfallspezifikation, Testung (JUnit) Bereitstellung API (Javadoc)	

Abgabe der Projektdokumentation	02.07.2016
Vorbereitung Präsentation	

Praesentation	23.06.2016

2.2 Ressourcenplanung und Organisation

Die Probleme bei der Entwicklung der Software ließen sich in drei große Bereiche aufteilen. Für die Kommunikation zwischen Server und Client war Dennis Hinterwimmer zuständig, um die visuelle Darstellung und das Userinterface kümmerte sich Paul Wolff und für die Verarbeitung der Daten

und das Berechnen der Routen übernahm Maxim Aisel. Besonders bei der Übergabe der Daten an die nächste Schicht, ergaben sich inhaltliche Schnittstellen. Zudem war es wichtig, die anderen Schichten zu kennen und so ergab sich des öfteren eine rollenübergreifende Zusammenarbeit.

2.3 Werkzeugunterstützung

2.3.1 Managementwerkzeuge

Um die verschiedenen Phasen der Entwicklung zu koordinieren, wurde die Versionskontrollsoftware Apache Subversion eingesetzt. Diese ermöglicht es im Team an Projekten zu arbeiten und die verschiedenen Änderungen an allen beteiligten Arbeitscomputern zusammenzufügen.

Zum Zwecke der Beschreibung und Dokumentation des Projektes wurde die Textverarbeitungssoftware Lyx genutzt, welche es ermöglicht gut strukturierte Dokumente zu erzeugen. Die Software übernimmt dabei den großteil der Formatierung.

Die Dokumentation des Quelltextes wurde mit Javadocs erzeugt. Javadocs liest spezielle Kommentare aus dem Quelltext und erzeugt daraus eine Html-Dokumentationsdatei.

Für Aufgabenverteilung und Kontrolle des Fortschritts wurde Trello (eine web-basierte Projektmanagementsoftware) eingesetzt.

2.3.2 Softwareentwicklungswerkzeuge

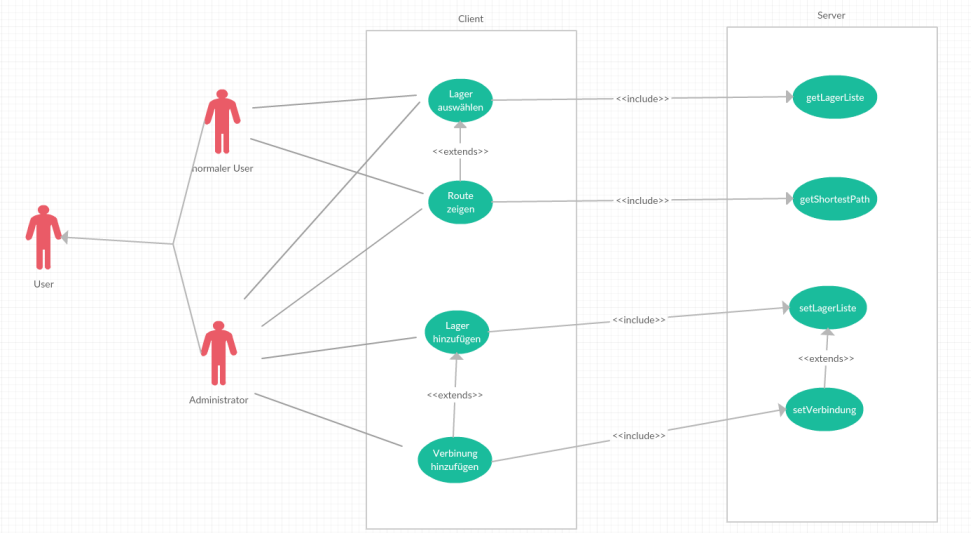
Für die Softwareentwicklung wurde die Entwicklungsumgebung Eclipse in Verbindung mit dem Java Development Kit verwendet. Diese ermöglichte das kompilieren der Quelltexte sowie die Fehleranalyse. Zur Erstellung von UMLs wurde Papyrus, ObjectAid und Creatly.com eingesetzt.

3 Pflichtenheft

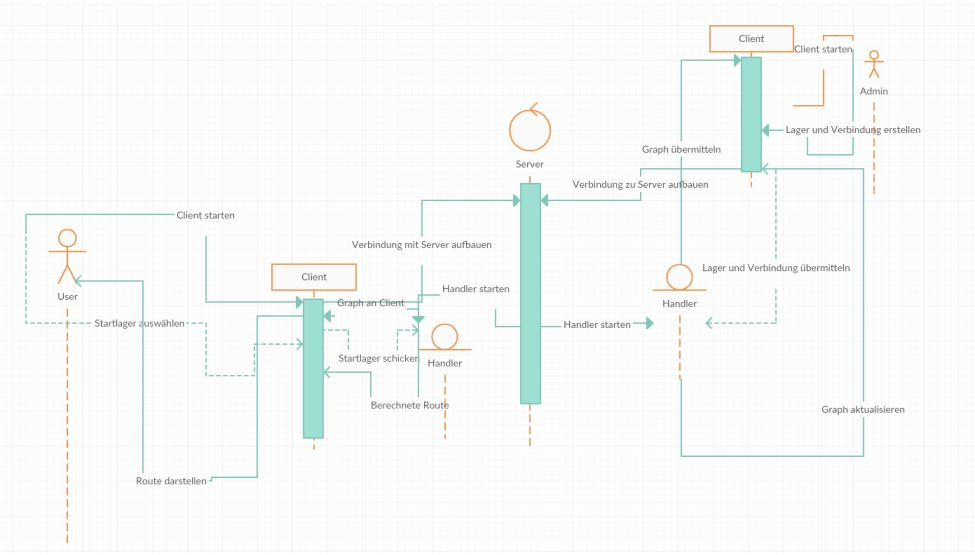
Das Pflichtenheft liegt gesondert bei.

4 Anforderungsanalyse und Entwurf

4.1 Anwendungsfallanalyse



Der normale User verfügt über die Funktionen zum Auswählen eines Lagers und zum Anzeigen einer Route. Der Administrator verfügt zusätzlich über die Funktionalitäten zum Hinzufügen eines Lagers und/oder einer Verbindung.



Der Server läuft in Schleife und startet pro verbundenen Client einen Handler, welcher die konkreten Anfragen behandelt und entsprechende Ergebnisse an den Client weiterleitet. Beispiel:

Der User startet den Client, welcher eine Verbindung mit dem Server auf-

baut. Der Server übergibt die Verbindung zum Client an einen Handler, der Handler liefert dem Client einen Graphen mit allen Lagern und Verbindungen.

Vom Userinterface (Client) werden diese Lager angezeigt und der User kann einen Startknoten auswählen. Der Startknoten wir mit dem Befehl zum berechnen einer Route an den Handler übermittelt. Dieser übernimmt nun die Berechnung und sendet das Ergebnis, die berechnete Route, an den Client zurück,

welcher die Daten für den User aufbereitet darstellt. Um überhaupt Lager zu besitzen bedarf es einen Administrator, diese legt er ebenfalls mit einem Clienttool an. Der Client wird wieder gestartet und einem Handler zugewiesen, der Client bekommt die bereits vorhandenen Lager und Verbindungen zur

Darstellung. Der Administrator (Admin) kann nun ein Lager oder eine Verbindung erstellen, welche an den Handler mit einem Befehl zum Erstellen desjeweiligen (Lager oder Verbindung) geschickt wird. Der Handler führt die Befehle aus und aktualisiert den Graphen auf der Clientseite des Administrators.

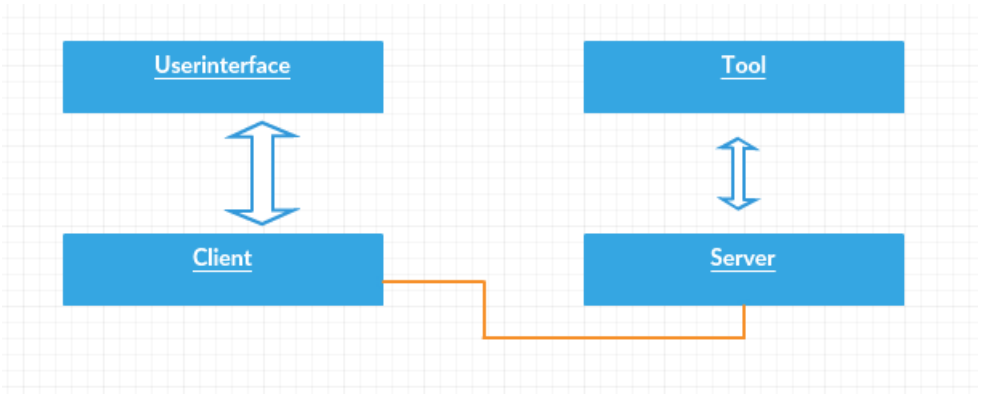
4.2 Problembereichsanalyse

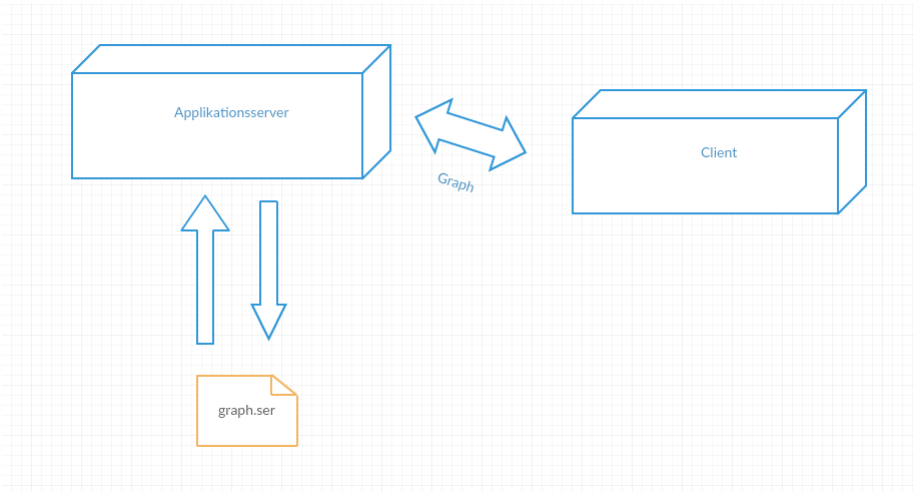
Um Routen zu berechnen und dem Nutzer darzustellen, benötigt man die Speicherung von Daten, die Berechnung der Routen, die Übertragung der Eingaben und die Eingaben selbst. Hierfür benötigt man also eine Klasse zum Brechnen, lesen und schreiben der Daten, eine Klasse zum Übertragen und eine Klasse zum Erstellen und Anfragen von Daten. Zudem ergeben sich Teilprobleme, wie der mehrfache Zugriff von Nutzern, das Übertragen der Daten zwischen den verschiedenen Klassen, die dauerhafte Speicherung und das parallele lesen und schreiben der Daten.

4.3 Stand der Wissenschaft und Technik

Da es sich um eine recht einfache Interaktion handelt, schien das Einarbeiten in Fremdsoftware/Fremdcode größtenteils unnötig. Lediglich die Erstellung von Graphen als Objekte eforderte Graphenklassen von Fremdanbietern.

4.4 Entwurf der Systemarchitektur

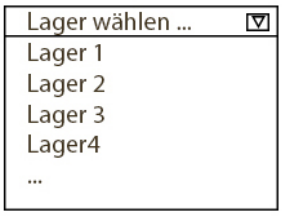




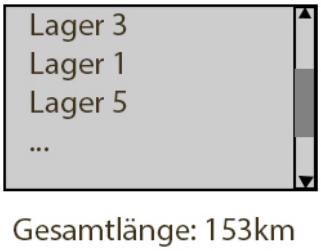
4.5 Entwurf der Benutzeroberfläche.

4.5.1 Entwürfe

4.5.1.1 Eingabe



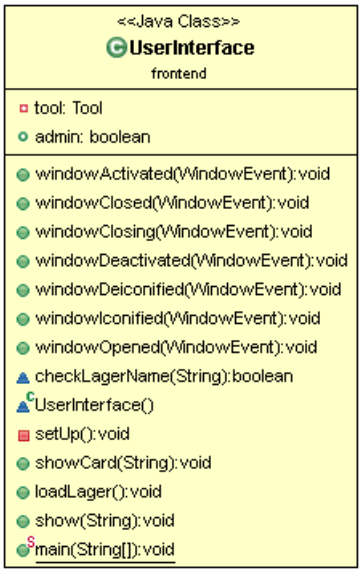
4.5.1.2 Ausgabe



4.5.2 Klassen

4.5.2.1 Launcher Die Startklasse, in der über jeweils einen Button entschieden werden kann, welches UserInterface gestartet werden soll. Dabei wird eine Instanz der jeweiligen Version des UserInteraces erstellt. Eine Checkbox setzt zusätzlich die Admin-Rechte.

4.5.2.2 UserInterface



Methode setUp() Erstellt die grafische Oberfläche und setzt die Komponentenzuweisungen bzw. Layouts

Methode showCard(String) Zeigt die als String übergebene Karte im Cardlayout

Methode loadLager() Holt die Lagerliste als String[] vom Clinet bzw. Tool und schreibt alle Lager in die DropDown Menüs.

Methode show(String) Holt die Route vom als String übergebenen Startlager vom Clienten bzw. Tool und speichert sie in ein Array.

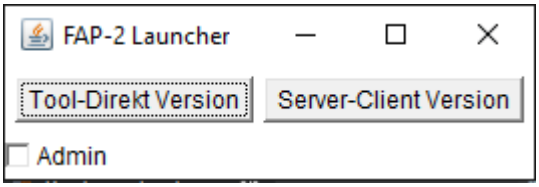
4.5.2.3 UserInterfaceClient

Ist identisch mit dem Tool-direkt UserInterface, ruft die Funktionen jediglich auf dem Client anstatt direkt auf Tool auf.

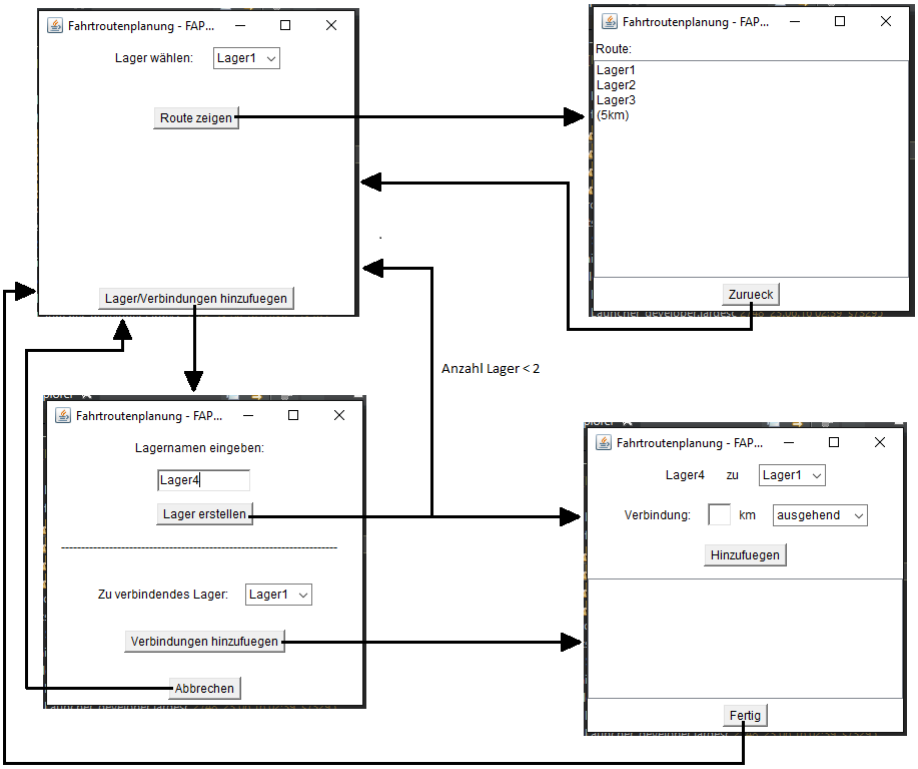
4.5.3 Navigation

4.5.3.1 Launcher

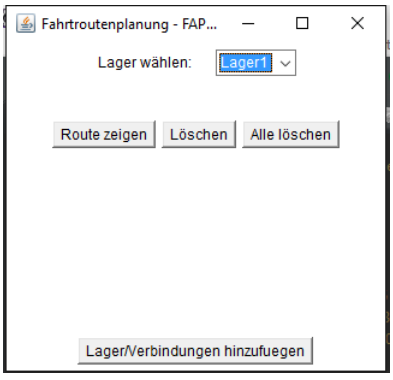
Der Launcher besteht aus 2 Buttons und einer Checkbox. Die Buttons führen zu den zwei verschiedenen Versionen des UserInterfaces. Die Checkbox dient dazu, Admin-Rechte in der Tool-direkt Version zu setzen. Sobald eine Version gewählt wurde, verschwindet der Launcher.



4.5.3.2 Menüführung im Userinterface

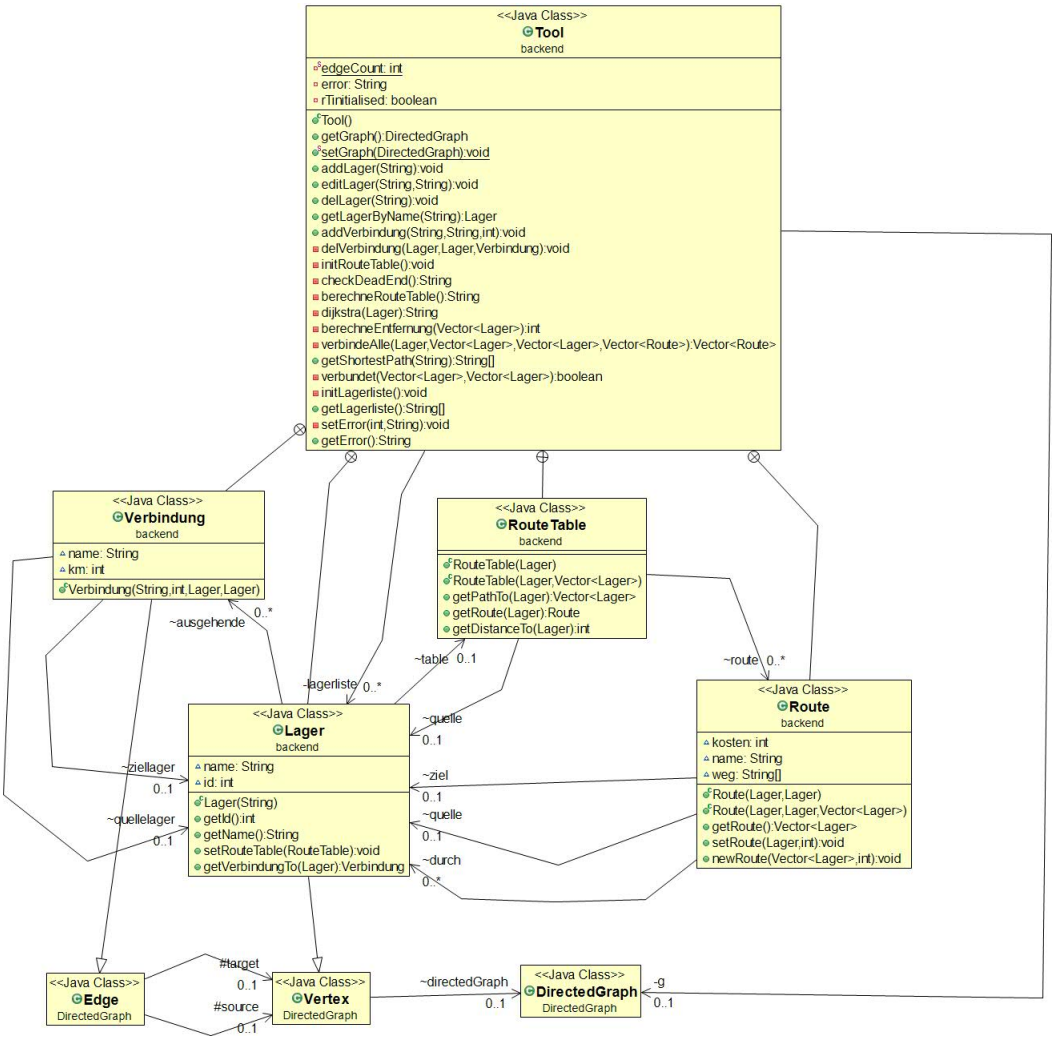


4.5.3.3 Adminfunktionen Gesetzte Adminrechte im Launcher erweitern das Tool-direkt UserInterface um einen „Löschen" und einen „Alle löschen" Button. „Löschen" entfernt lediglich das gewählte Lager und alle Verbindungen von und zu diesem. „Alle löschen" entfernt alle Lager.



5 Entwurf der Funktionalität/Interaktionsmodell

5.1 Tool



Methode addLager(String) Fügt einen neuen Lager zu dem Graphen hinzu

Methode editLager(String, String) Ändert Lagerbezeihnung

Methode delLager(String) Löscht einen Lager sowie seine eingehende und ausgehende Verbindungen

Methode getLagerByName(String) Liefert einen Lager aus Lagerliste zurück

Methode addVerbindung(String, String , int) Fügt eine neue Verbindung zwischen zwei Lager im Graphen hinzu. Falls die Verbindung zwischen Quelllager und Ziellagern schon existiert wird sie überschrieben.

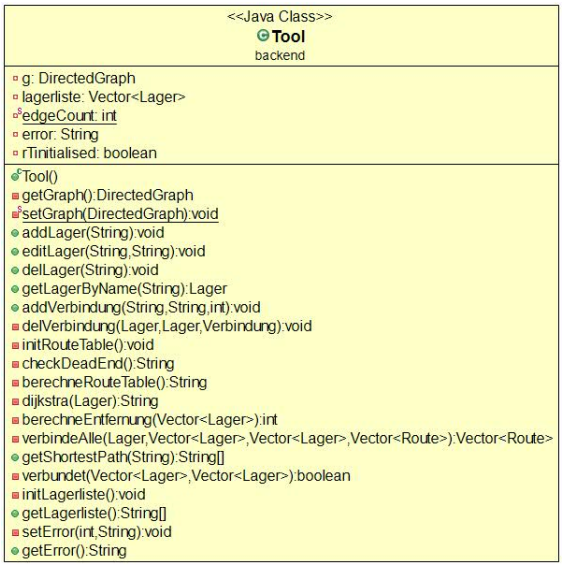
Methode getShortestPath(String) Liefert kürzeste Route, die alle Lager ausgehend vom Startlager verbindet, zurück.

Methode getLagerliste() Liefert ein Liste mit den Bezeichnungen von allen Lagern

Methode getError() Liefert aktuelle Fehlermeldung zurück.

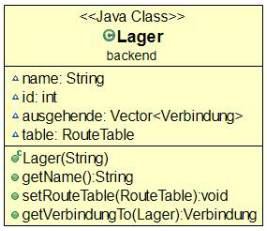
5.1.1 Klasse Tool

- verwaltet die Fahrtroutenplanung



5.1.1.1 Klasse Lager

- stellt einen Knoten im Graph dar
- verwaltet Lager.

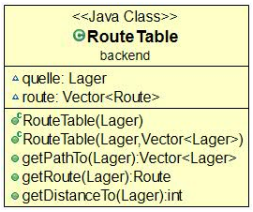


5.1.1.2 Klasse Verbindung

- stellt eine Kante im Graph dar
- verwaltet Verbindungen.

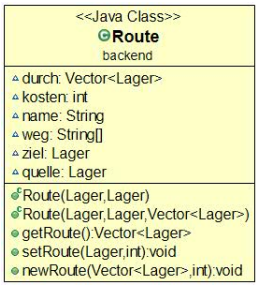
5.1.1.3 Klasse RouteTable

- verwaltet alle Strecken eines Lagers



5.1.1.4 Innere Klasse Route

- verwaltet eine Strecke



6 Datenverwaltung / Datenbankentwurf

Die Datenspeicherung beschränkt sich auf ein serialisiertes Objekt der Klasse Graph. Das serialisierte Objekt wird im Ordner indem der Server gestartet wird als graph.ser abgespeichert. Bei der Tool-direkt Version des UserInterfaces erfolgt die serialisierung im Ordner des Launchers.

7 Implementation

7.1 Wichtigste Algorithmen der Klasse Tool

7.1.1 Dijkstra-Algorithmus

- Anpassung von Dijkstra-Algorithmus
- berechnet einen kürzesten Pfad zwischen dem gegebenen Startlager und übrigen Lagern

```
FUNKTION dijkstra(startKnoten)
START
    FÜR i = 0,..,KnotenAnzahl TUE
        AbstandZumStartknoten[i] = unendlich
        Vorgänger[i] = NULL
    AbstandZumStartknoten[startKnoten] = 0
    Vorgänger[startKnoten] = startKnoten
    warteschlange.ADD(startKnoten)

    SOLANGE warteschlange NICHT leer
        aktuellerKnoten = warteschlange.erstesElement()

        FÜR JEDEN Nachbarknoten AUS aktuellerKnoten  TUE
            Distanz = AbstandZumStartknoten[aktuellerKnoten] + KostenDerKante

            WENN warteschlange.contains[Nachbarknoten] UND  = AbstandZumStartknoten[Nachbarknoten] > Distanz DANN
                AbstandZumStartknoten[Nachbarknoten] = Distanz
                Vorgänger[Nachbarknoten] = aktuellerKnoten

            WENN Vorgänger[Nachbarknoten] == NULL DANN
                AbstandZumStartknoten[Nachbarknoten] = Distanz
                Vorgänger[Nachbarknoten] = aktuellerKnoten
                warteschlange.ADD(Nachbarknoten)

    liste = lagerliste
    liste.REMOVE(startKnoten)

    FÜR JEDEN Knoten AUS liste
        tmp = Vorgänger[Knoten]
        WENN tmp == NULL DANN
            error
        SONST
            SOLANGE tmp != startKnoten TUE
                hilfsliste.AddAtFistPosition(tmp)
                tmp = Vorgänger[tmp]

            r = RouteZum(Knoten)
            r.newRoute(hilfsliste,AbstandZumStartknoten[Knoten])
            hilfsliste.clear()
ENDE
```

7.1.2 VerbindeAlle -Algorithmus

- rekursiver Algorithmus
- findet alle mögliche Routen, die alle Lager verbinden

```
FUNKTION verbindeAlle(startLager, abgearbeiteteLager[], nichtAbgearbeiteteLager[], RoutenDurchAlleLager[])
START
    done[] = abgearbeiteteLager[];
    WENN done == leer ODER done.lastElement() != startLager DANN
        done.addElement(startLager);
    toDoListe[] = nichtAbgearbeiteteLager[];
    toDoListe.entferne(startLager);

    WENN RouteDurchAlleLager == NULL DANN
        RoutenDurchAlleLager[] = new RoutenDurchAlleLager[]

    WENN toDoListe.size() == 0  DANN
        zwischenLager[] = done(ohne erstes und letztes Element)
        neuRoute = newRoute(done.lastElement(), done.firstElement(), zwischenLager)
        neueRoute.kosten = berechneEntfernung(done);
        RoutenDurchAlleLager.addElement(neueRoute);

    WENN (toDoListe != leer UND verbundet(lagerliste, done) == false) DANN
        FÜR JEDEN Lager AUS toDoListe TUE
            tmpListe[] = startLager.table.getPathTo(lager)
            done[] += tmpListe[]
            verbindeAlle(Lager, done[], toDoListe[], RoutenDurchAlleLager[])

            FÜR i = tmpListe.size(),.., i > 0 TUE
                done.löscheLetztesElement()
            tmpListe.clear()

    RETURN RoutenDurchAlleLager[];
ENDE
```

7.1.3 getShortestPath -Algorithmus

- Wählt kürzeste Route aus
- bekommt Routen zur Auswahl vom VerbindeAlle -Algorithmus

```
FUNKTION getShortestPath(Startlager)
START
    WENN rTinitialised == false DANN
        initRouteTable();

    WENN (error == null && rTinitialised) DANN
        entfernung = Integer.MAX_VALUE;
        RoutenDurchAlleLager[] = new RoutenDurchAlleLager[]

        RoutenDurchAlleLager[] = verbindeAlle(Startlager, leereListe[], alleLager[], null)
            FÜR JEDE Route AUS RoutenDurchAlleLager[] TUE
                WENN Route.kosten < entfernung DANN
                    entfernung = Route.kosten;
                    kürzesteRoute = Route;

            ShortestPath[] = new String[kürzesteRoute.weg.length + 2];
            ShortestPath[0] = kürzesteRoute.quelle.name;

            FÜR i = 0,..., i < kürzesteRoute.weg.length TUE
                ShortestPath[i + 1] = kürzesteRoute.weg[i];

            ShortestPath[letztesElement] = kürzesteRoute.kosten
            return ShortestPath;
    SONST
        RETURN null;
ENDE
```

7.2 Der Build-Prozess

7.2.1 Build

Es werden zwei Programme ausgeliefert, einmal das Client-Programm mit GUI bzw. der Launcher der zwischen Server-Client Version und Tool-direkt Version differenziert, und dem Server Programm. Beide werden dem Kunden als lauffähige Java Jar Datei ausgeliefert und können auf unterschiedlichen Rechnern ausgeführt werden.

Des Weiteren wird eine Entwickler-Version der beiden genannten Programme ausgeliefert, die den Quellcode für eine mögliche Weiterentwicklung enthalten.

Alle builds wurden mit dem Eclipse JAR Export erstellt. Im Ordner builds des Projekts ist darüber hinaus jeweils eine .jardesc zu finden, die die Einstellungen der einzelnen builds enthält und mit der ggf. die einzelnen Programmteile erneut erstellt werden können.

7.2.2 Installation

Zur Ausführung der Programme ist es nötig, dass Java 1.8 oder neuer auf den zu verwendenden Rechnern installiert ist.

Launcher.jar (Gui/Client) kann von jedem Ort/Rechner durch einfaches doppeltes Klicken ausgeführt werden. Zur Nutzung der Server-Client Version ist es jedoch nötig, dass der Server zuvor gestartet wurde. Die Tool-direkt Version erfordert keine weiteren Schritte und ist ohne Server lauffähig.

Server.jar sollte wenn möglich über die Eingabeaufforderung(Windows) bzw. das Terminal (Mac) mitttels java -jar Server.jar gestartet werden, damit die Ausgaben des Servers in der Konsole sichtbar werden. Der Server kann aber

auch durch einfaches doppeltes Klicken gestartet werden, dann sind jedoch keine Ausgaben des Server, inklusive Fehlermeldungen, sichtbar.

Außerdem sollten Server und Client in einem internen gemeinsamen Netz laufen, da die IP-Adresse des Servers im Moment fest im Code des Clients programmiert ist. Die gewünschte IP-Adresse ist den Entwicklern vor der Auslieferung mitzuteilen.

7.3 API-Dokumentation

Die API-Dokumentation wurde mithilfe von Eclipse erstellt. Hierfür wurde lediglich eine spezielle Kommentarform gewählt, die die Pararmeter und Rückgabewerte für die Dokumentation deklariert. Anschließend konnte die API-Dokumentation generiert werden. Diese Dokumentation liegt dem Ordner Doc bei.

7.4 Teststrategien und -werkzeuge

7.4.1 Tool

Testgegenstand:	Klasse Tool
Wieso?	Als Basiskomponente die in Anwendung verwendet wird, muss Klasse Tool grundstabil sein.
Wann?	Nach der der Entwicklung der Klasse
Wo?	An beliebigem Computer, weil keine Datenbankverbindung oder web-service für die Klasse notwendig sind
Wie?	Basis-Instrument: Unit Tests. Manuelle Tests: Beispielanwendungen und Dummy-Anwendung konstruiren und verschieden Szenarien durchspielen, Risiko-basierte Analyse zum Finden von Bugs
Womit?	JUnit Testing Framework Version 4.12.0
Wer?	Entwickler der Klasse
Akzeptanzkriterien:	Umsetzung der Funktionen durch ausreichend viele Unit Tests bestätigen. Solides und absolut stabiles Verhalten in den manuellen Tests. Keine Abstürtze in normalen Verwendungsszenarien.

7.4.2 UI

Testgegenstand:	Klassen UserInterface, UserInterfaceClient, Launcher
Wieso?	Da dies Klassen alles sichtbaren Elemente und Ausgaben verwalten, ist es wichtig dass sie möglichst Fehlerunanfällig sind.
Wann?	Während der Entwicklung der Klassen
Wo?	An beliebigem Computer. Zum Testen der Client Version des UserInterfaces wurde der Server localhost oder auf einem zweiten Rechner gestartet.
Wie?	Das UserInterface wurde manuell getestet, indem alle Funktionen einzeln ausgeführt wurden.
Womit?	Es waren keine weiteren Tools notwendig
Wer?	Entwickler der Klassen
Akzeptanzkriterien:	Stabile Programmnutzung ohne Abstürze. Fehlerausgabe bei Fehlern auf Server/Client oder bei falschen Eingaben.

7.5 Testdurchführung und Testergebnisse

7.5.1 Algemeine Beschreibung

1)	Vorbedingungen die vor der Testausführung hergestellt werden müssen
2)	die Benennung des Testobjekts und der Spezifikationen, auf die sich der Testfall bezieht
3)	die Eingabedaten für die Durchführung des Tests
4)	die Handlungen, die zur Durchführung des Testfalls notwendig sind
5)	die erwarteten Ergebnisse und / oder Reaktionen des Testobjektes auf die Eingaben
6)	die erwarteten Nachbedingungen, die als Ergebnis der Durchführung des Testfalls erzielt werden
7)	die Prüfanweisungen, d. h. wie Eingaben an das Testobjekt zu übergeben sind und wie Sollwerte abzulesen sind

7.5.2 testTool()

1.	1)	Lagerliste muss leer sein
	2)	tool.tool() Konstruktor der Klasse
	3)	-
	4)	neue Instanz der Klasse Tool t1 erzeugen, Lagerliste holen
	5)	Lagerliste ist leer, weil kein Lager im Graph existiert
	6)	-
	7)	assertEquals(0, t.getLagerliste());

2.	1)	Lagerliste muss leer sein
	2)	tool.tool() Konstruktor der Klasse
	3)	
	4)	Lager erstellen, zweite Instanz der Klasse Tool t2 erzeugen
	5)	t2 sieht unter t1 erstellten Lager
	6)	Ein Lager wurde erstellt
	7)	assertEquals(t1.getLagerliste().length, t2.getLagerliste().length);

7.5.3 testAddLager()

1.	1)	Lagerliste muss leer sein
	2)	tool.addLager(String name)
	3)	name = ”test”
	4)	neue Instanz der Klasse Tool t erzeugen, Lager mit dem Namen ”test” erstellen
	5)	Ein Lager wird erstellt und aus dem Graphen geholt, es kommt keine Fehlermeldung
	6)	Ein Lager wurde erstellt
	7)	assertNull(t.getError()); assertEquals(”test”, t.getLagerByName(”test”).name); assertEquals(t.getLagerByName(”test”).id, t.getLagerliste().length);

2.	1)	Lager mit dem Namen ”test”
	2)	tool.addLager(String name)
	3)	name = ”test”
	4)	Lager mit einem bereits existierenden Namen erstellen
	5)	Lagerliste bleibt unverändert, es lieigt eine Fehlermeldung vor
	6)	-
	7)	assertEquals(1, t.getLagerliste().length); assertNotNull(t.getError());

7.5.4 testEditLager()

1.	1)	Lagerliste muss leer sein
	2)	tool.editLager(String name, String newName)
	3)	name = ”alt”, newName = ”neu”
	4)	neue Instanz der Klasse Tool t erzeugen, Lager mit dem Namen ”alt” erstellen und in ”neu” umbennenen
	5)	Lager ”alt” wurde in ”neu” umbenannt, es kommt keine Fehlermeldung, Lagerliste bleibt unverändert, der Lager ist unter altem Namen nicht mehr im Graphen zu finden
	6)	
	7)	assertNull(t.getError()); assertEquals(”neu”, t.getLagerByName(”neu”).name); assertEquals(1, t.getLagerliste().length); assertNull(t.getLagerByName(”alt”));

7.5.5 testDelLager()

1.	1)	Lagerliste muss leer sein
	2)	tool.delLager(String name)
	3)	name = ”test”
	4)	neue Instanz der Klasse Tool t erzeugen, Lager mit dem Namen ”test” erstellen, Lager ”test” löschen
	5)	Ein Lager mit dem Namen=”test” löschen
	6)	Ein Lager wurde gelöscht, Lagerliste ist leer
	7)	assertNull(t.getError()); assertNull(t.getLagerByName(”test”)); assertEquals(0, t.getLagerliste().length);

2.	1)	Lagerliste muss leer sein
	2)	tool.delLager(String name)
	3)	name = ”test”
	4)	Lager mit dem Namen ”test” löschen
	5)	es lieigt eine Fehlermeldung vor
	6)	
	7)	assertNotNull(t.getError());

7.5.6 testGetLagerByName()

1.	1)	Lagerliste muss leer sein
	2)	tool.testGetLagerByName(String name)
	3)	name = ”test”
	4)	neue Instanz der Klasse Tool t erzeugen, Lager mit dem Namen ”test” aus dem Graphen holen
	5)	Lager wird im Graphen nicht gefunden
	6)	
	7)	assertNull(t.getLagerByName(”test”));

2.	1)	Lagerliste muss leer sein
	2)	tool.testGetLagerByName(String name)
	3)	name = ”test”
	4)	Lager mit dem Namen ”test” und ”test2” erstllen
	5)	zwei unterschiedliche Lager befinden sich im Graphen
	6)	Lager mit dem Namen ”test” wurde erstellt; Lager mit dem Namen ”tes2t” wurde erstellt
	7)	assertSame(t.getLagerByName(”test”), t.getLagerByName(”test”)); assertNotSame(t.getLagerByName(”test”), t.getLagerByName(”test2”));

7.5.7 testAddVerbindung()

1.	1)	Lagerliste muss leer sein
	2)	tool.addVerbindung(String out, String in, int km);
	3)	out=”A”, in=”B”,km=1
	4)	neue Instanz der Klasse Tool t erzeugen; neue Verbinung von ”A” nach ”B” erstellen
	5)	es kommt eine Fehlermeldung, weil beide Lager nicht existieren
	6)	
	7)	assertNotNull(t.getError());

2.

1)	Lagerliste muss leer sein
2)	tool.addVerbindung(String out, String in, int km);
3)	out=”A”, in=”B”,km=1
4)	Lager mit dem Namen ”test” erstellen; neue Verbinung von ”A” nach ”B” erstellen
5)	es kommt eine Fehlermeldungm, weil 2.Lager nicht existiert
6)	Lager mit dem Namen ”A” wurde erstellt
7)	assertNotNull(t.getError());

3.

1)	Lager mit dem Namen ”A”
2)	tool.addVerbindung(String out, String in, int km);
3)	out=”A”, in=”B”,km=0
4)	Lager mit dem Namen ”B” erstellen, neue Verbinung von ”A” nach ”B” erstellen
5)	es kommt eine Fehlermeldungm, weil Distanz kleiner als 1 ist
6)	Lager mit dem Namen ”B” wurde erstellt
7)	assertNotNull(t.getError());

4.

1)	Lager mit dem Namen ”A”, Lager mit dem Namen ”B”
2)	tool.addVerbindung(String out, String in, int km);
3)	out=”A”, in=”B”,km=1
4)	neue Verbinung von ”A” nach ”B” erstellen
5)	Verbindung von ”A” nach ”B” wurde erstellt
6)	Verbindung von ”A” nach ”B” befindet sich im Graphen
7)	assertEquals(1, t.getLagerByName(”A”).ausgehende.size()); assertEquals(”B”, t.getLagerByName(”A”). ausgehende.firstElement().ziellager.name); assertEquals(”A”, t.getLagerByName (”A”).ausgehende.firstElement().quellelager.name); assertEquals(0, t.getLagerByName(”B”).ausgehende.size());

5.	1)	Lager mit dem Namen ”A”, Lager mit dem Namen ”B”
	2)	tool.addVerbindung(String out, String in, int km);
	3)	out=”A”, in=”B”,km=2
	4)	Eine existierend Verbindung wird überschrieben
	5)	neue Distanz von ”A” nach ”B” ist 2 km
	6)	
	7)	assertEquals(1, t.getLagerByName(”A”).ausgehende.size()); assertEquals(2, t.getLagerByName (”A”).getVerbindungTo(t.getLagerByName(”B”)).km);

7.5.8 testGetShortestPath()

1.	1)	Lagerliste muss leer sein
	2)	tool.getShortestPath(String name)
	3)	name=”A”
	4)	neue Instanz der Klasse Tool t erzeugen; ein Lager mit dem Namen ”A” erzeugen; kürzeste Route ausgehend vom ”A” berechnen
	5)	kürzeste Route wurde nicht berechnen; es kommt eine Fehlermeldung,weil Lagerliste weniger als 2 Lager hat
	6)	ein Lager mit dem Namen ”A” wurde erzeugt
	7)	assertNotNull(t.getError());

2.	1)	Lager mit dem Namen ”A”
	2)	tool.getShortestPath(String name)
	3)	name=”A”
	4)	ein Lager mit dem Namen ”B” erstellen; kürzeste Route ausgehend vom ”A” berechnen
	5)	kürzeste Route wurde nicht berechnen; es kommt eine Fehlermeldung, weil keine Verbindung zwischen A und B existiert
	6)	ein Lager mit dem Namen ”B” wurde erstellt
	7)	assertNotNull(t.getError());

3.

1)	Lager mit dem Namen ”A”, Lager mit dem Namen ”B”
2)	tool.getShortestPath(String name)
3)	name=”A”
4)	eine Verbinung von ”A” nach ”B” erstellen; kürzeste Route ausgehend vom ”A” berechnen
5)	kürzeste Route wurde nicht berechnen; es kommt eine Fehlermeldung, weil Lager ”B” eine Sackgasse ist
6)	ine Verbinung von ”A” nach ”B” wurde erstellen
7)	assertNotNull(t.getError());

4.

1)	Lager mit dem Namen ”A”, Lager mit dem Namen ”B”, Verbinung von ”A” nach ”B”
2)	tool.getShortestPath(String name)
3)	name=”A”
4)	eine Verbinung von ”B” nach ”A” erstellen, kürzeste Route ausgehend vom ”A” berechnen
5)	Findet den kürzesten Weg ausgehend von ”A”
6)	Verbinung von ”B” nach ”A” wurde erstellt
7)	assertNull(t.getError()); assertEquals(3, t.getShortestPath(”A”)); assertEquals(”A”, path[0]); assertEquals(”B”, path[1]); assertEquals(”(1km)”, path[2]);

5.

1)	Lager mit dem Namen ”A”, Lager mit dem Namen ”B”, Verbinung von ”A” nach ”B” Verbinung von ”B” nach ”A”
2)	tool.getShortestPath(String name)
3)	name=”A”
4)	ein Lager mit dem Namen ”C” erstellen, ein Lager mit dem Namen ”D” erstellen, eine Verbinung von ”C” nach ”D” erstellen, eine Verbinung von ”D” nach ”C” erstellen kürzeste Route ausgehend vom ”A” berechnen
5)	Lager mit dem Namen ”C” wurde erstellt, Lager mit dem Namen ”D” erstellt, kürzeste Route wurde nicht berechnen; es kommt eine Fehlermeldung, weil keine Verbingung von A-B nach C-D existiert
6)	Verbinung von ”C” nach ”D” wurde erstellt, Verbinung von ”D” nach ”C” wurde erstellt
7)	assertNull(t.getShortestPath(”A”)); assertNotNull(t.getError());

6.	1)	Lager mit dem Namen ”A”, Lager mit dem Namen ”B”, Lager mit dem Namen ”C”, Lager mit dem Namen ”D”, Verbinung von ”A” nach ”B” Verbinung von ”B” nach ”A” Verbinung von ”C” nach ”D” Verbinung von ”D” nach ”C”
	2)	tool.getShortestPath(String name)
	3)	name=”A”
	4)	eine Verbinung von ”B” nach ”C” erstellen, eine Verbinung von ”C” nach ”B” erstellen, kürzeste Route ausgehend vom ”A” berechnen
	5)	Findet den kürzesten Weg ausgehend von ”A”
	6)	
	7)	assertNull(t.getError());

7.	1)	Lagerliste muss leer sein; String[] test ={”A”,”B”,”E”,”F”,”D”,”C”,”(43km)”};
	2)	tool.getShortestPath(String name)
	3)	name=”A”
	4)	6 Lager mit dem Namen: ”A”,”B”,”C”,”D”,”E”,”F” erstellen, Verbinungen: von ”A” nach ”B” = 10km , von ”A” nach ”C” = 20km, von ”B” nach ”A” = 10km, von ”B” nach ”E” = 10km, von ”B” nach ”D” = 50km, von ”C” nach ”A” = 20km, von ”C” nach ”D” = 20km, von ”C” nach ”E” = 33km, von ”D” nach ”B” = 50km, von ”D” nach ”C” = 20km, von ”D” nach ”E” = 20km, von ”D” nach ”F” = 2km, von ”E” nach ”B” = 10km, von ”E” nach ”C” = 33km, von ”E” nach ”D” = 20km, von ”E” nach ”F” = 1km, von ”F” nach ”D” = 2km, von ”F” nach ”E” = 1km erstellen; kürzeste Route ausgehend vom ”A” berechnen
	5)	Findet den kürzesten Weg ausgehend von ”A”
	6)	Alle Lager und Verbundungen wurde erstellt
	7)	String[] shortetestPath = t.getShortestPath(”A”); for(int i =0; i< shortetestPath.length;i++) assertEquals(test[i],shortetestPath[i]);

7.5.9 testGetLagerliste()

1.

1)	Lagerliste muss leer sein
2)	tool.getLagerliste()
3)	
4)	neue Instanz der Klasse Tool t erzeugen; Lagerliste holen
5)	Lagerliste ist leer, weil kein Lager im Graph existiert
6)	
7)	assertEquals(0, t.getLagerliste().length);

2.

1)	Lagerliste muss leer sein
2)	tool.getLagerliste()
3)	
4)	Lager mit dem Namen ”test1” erstellen, Lager mit dem Namen ”test2” erstellen, Lagerliste holen
5)	neu ertellte Lagern sind in der Lagerliste
6)	zwei Lager wurden erstellt
7)	assertEquals(2, t.getLagerliste().length);

7.5.10 testGetError()

1.

1)	Lagerliste muss leer sein
2)	tool.getError()
3)	
4)	neue Instanz der Klasse Tool t erzeugen, Fehlermeldung holen
5)	Fehlermeldung ist NULL solange kein Fehler kommt
6)	
7)	assertNull(t.getError());

2.

1)	Lagerliste muss leer sein
2)	tool.getError()
3)	
4)	Lager löschen, Fehlermeldung holen
5)	es liegt eine Fehlermeldung vor
6)	
7)	assertEquals(”Lager: test wurde nicht gefunden”, t.getError());

3.

1)	Lagerliste muss leer sein
2)	tool.getError()
3)	
4)	Lagerbezeichnung ändern, Fehlermeldung holen
5)	es liegt eine Fehlermeldung vor
6)	
7)	assertEquals(”Lager: test wurde nicht gefunden”, t.getError());

4.

1)	Lagerliste muss leer sein
2)	tool.getError()
3)	
4)	kürzeste Route ausgehend vom ”test” berechnen, Fehlermeldung holen
5)	es liegt eine Fehlermeldung vor
6)	
7)	assertEquals(”Lagerliste hat weniger als 2 Lager”, t.getError());

5.

1)	Lagerliste muss leer sein
2)	tool.getError()
3)	
4)	Lager mit dem Namen ”test” erstellen, Lager mit dem Namen ”test” erstellen, Fehlermeldung holen
5)	es liegt eine Fehlermeldung vor
6)	ein Lager mit dem Namen ”test” erstellt
7)	assertEquals(”Lager: test exitiert schon”, t.getError());

6.

1)	Lager mit dem Namen ”test”
2)	tool.getError()
3)	
4)	Verbindung von ”test” nach ”test2” erstellen, Fehlermeldung holen
5)	es liegt eine Fehlermeldung vor
6)	
7)	assertEquals(”Lager: test2 wurde nicht gefunden”, t.getError());

7.

1)	Lager mit dem Namen ”test”
2)	tool.getError()
3)	
4)	Verbindung von ”test2” nach ”test” erstellen, Fehlermeldung holen
5)	es liegt eine Fehlermeldung vor
6)	
7)	assertEquals(”Lager: test2 wurde nicht gefunden”, t.getError());

8.

1)	Lager mit dem Namen ”test”
2)	tool.getError()
3)	
4)	Lager mit dem Namen ”test2” erstellen, Verbindung von ”test2” nach ”test” mit der Distanz=0 erstellen, Fehlermeldung holen
5)	es liegt eine Fehlermeldung vor
6)	Lager mit dem Namen ”test2” wurde erstellt
7)	assertEquals(”Disntanz in km muss größer als 0 sein”, t.getError());

9.

1)	Lager mit dem Namen ”test”
2)	Lager mit dem Namen ”test2”
3)	tool.getError()
4)	kürzeste Route ausgehend vom ”test” berechnen, Fehlermeldung holen
5)	es liegt eine Fehlermeldung vor
6)	
7)	assertEquals(”Lager: test ist eine Sackgasse ”, t.getError());

10.

1)	Lager mit dem Namen ”test” Lager mit dem Namen ”test2”
2)	tool.getError()
3)	
4)	Verbindung von ”test” nach ”test2” erstellen, kürzeste Route ausgehend vom ”test” berechnen, Fehlermeldung holen
5)	es liegt eine Fehlermeldung vor
6)	Verbindung von ”test” nach ”test2” wurde erstellt
7)	assertEquals(”Lager: test2 ist eine Sackgasse ”, t.getError());

11.	1)	Lager mit dem Namen ”test” Lager mit dem Namen ”test2” Verbindung von ”test” nach ”test2”
	2)	tool.getError()
	3)	
	4)	Verbindung von ”test2” nach ”test” erstellen, Lager mit dem Namen ”test3” erstellen, Verbindung von ”test3” nach ”test” erstellen, kürzeste Route ausgehend vom ”test” berechnen, Fehlermeldung holen
	5)	es liegt eine Fehlermeldung vor
	6)	erbindung von ”test2” nach ”test” wurde erstellt, Lager mit dem Namen ”test3” wurde erstellt, Verbindung von ”test3” nach ”test” wurde erstellt
	7)	assertEquals(”Lager: test3 hat zu wenig Verbindungen”, t.getError());

12.	1)	Lager mit dem Namen ”test” Lager mit dem Namen ”test2” Lager mit dem Namen ”test3” Verbindung von ”test” nach ”test2” Verbindung von ”test2” nach ”test” Verbindung von ”test3” nach ”test”
	2)	tool.getError()
	3)	
	4)	Verbindung von ”test2” nach ”test3” erstellen, kürzeste Route ausgehend vom ”test” berechnen, Fehlermeldung holen
	5)	es liegt keine Fehlermeldung vor
	6)	Verbindung von ”test2” nach ”test3” wurden erstellt
	7)	assertNull(t.getError());

7.6 Testdurchführung und Testergebnisse

7.6.1 Klasse Tool:

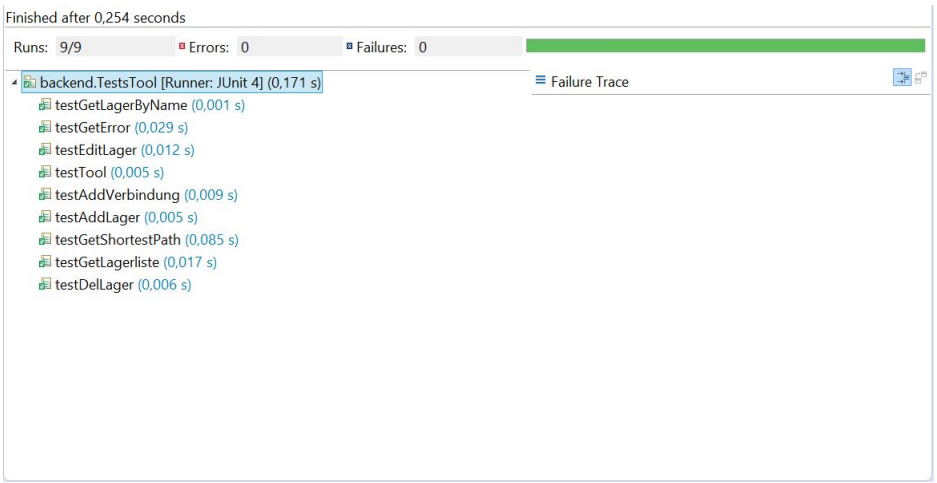
Testtiefe:	Mittel
Testfallermittlungsverfahren:	Kombination von Blackbox-Methode und intuitivem Testen

7.6.1.1 Testabdeckung Testabdeckung wurde mit Hilfe von Coverage Report ermittelt und beträgt 96%.

Tool

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
addLager(String)		100%		100%	0	2	0	12	0	1
addVerbindung(String, String, int)		100%		100%	0	6	0	18	0	1
berechneEntfernung(Vector)		100%		75%	1	3	0	7	0	1
berechneRouteTable()		100%		100%	0	3	0	6	0	1
checkDeadEnd()		100%		100%	0	3	0	7	0	1
delLager(String)		100%		100%	0	2	0	9	0	1
delVerbindung(Tool.Lager, Tool.Lager, Tool.Verbindung)		100%		100%	0	2	0	9	0	1
dijkstra(Tool.Lager)		100%		100%	0	10	0	39	0	1
editLager(String, String)		100%		100%	0	2	0	8	0	1
getError()		100%		n/a	0	1	0	1	0	1
getGraph()		68%		100%	0	2	5	19	0	1
getLagerByName(String)		100%		100%	0	3	0	6	0	1
getLagerliste()		100%		100%	0	2	0	4	0	1
getShortestPath(String)		100%		92%	1	7	0	21	0	1
initLagerliste()		100%		100%	0	2	0	6	0	1
initRouteTable()		100%		100%	0	5	0	13	0	1
setError(int, String)		100%		86%	1	7	0	14	0	1
setGraph(DirectedGraph)		72%		50%	1	2	3	17	0	1
static {...}		100%		n/a	0	1	0	1	0	1
Tool()		100%		100%	0	2	0	11	0	1
verbindeAlle(Tool.Lager, Vector, Vector, Vector)		100%		94%	1	9	0	25	0	1
verbundet(Vector, Vector)		100%		100%	0	2	0	3	0	1
Total	40 of 1,102	96%	5 of 107	95%	5	78	8	256	0	22

7.6.1.2 Testbericht Es wurden Tests für alle public-methoden der Klasse Tool implementiert. Public-methoden benutzten private-methoden der Klasse Tool, damit wurden durch Testfälle alle Methoden der Klasse ausgeführt. Alle Tests wurden bestanden.



7.6.2 Klassen Launcher bzw UserInterface und UserInterfaceClient

Testtiefe:	Mittel
Testfallermittlungsverfahren:	Intuitives Testen

7.6.2.1 Testabdeckung Testabdeckung kann nicht genau ermittelt werden, da die Funktionen manuell während der Entwicklung getestet wurden.

Bei der Tool-direkt Version wurden aber alle öffentlichen Methoden von Tool auf einer Instanz von Tool getestet.

Tool.getLagerliste()	no fails
Tool.addLager(String)	no fails
Tool.delLager(String)	no fails
Tool.addVerbindung(Lager, Lager, int)	no fails
Tool.getShortestPath(Lager)	no fails
Tool.getError()	no fails

Die Server-Client Version konnte nicht vollständig getestet werden, da bereits essentielle Funktionen wie das Hinzufügen von Lagern zu Fehlern führte.

7.6.2.2 Testbericht In der Tool-direkt Version konnten keine Fehler festgestellt werden.

Bei der Server-Client Version kam es jedoch zu Abstürzen und Einfrieren des Programmes, was auf fehlerhafte Server-Client Kommunikation zurückzuführen war. Die Ergebnisse varierten jedoch von Rechner zu Rechner. Bereits essentielle Funktionen wie das Hinzufügen von Lagern führten zu Fehlern.

8 Anwenderdokumentation

Zur Ausführung der Programme ist es nötig, dass Java 1.8 oder neuer auf den zu verwendenden Rechnern installiert ist.

Launcher.jar (Gui/Client) kann von jedem Ort/Rechner durch einfaches doppeltes Klicken ausgeführt werden. Zur Nutzung der Server-Client Version ist es jedoch nötig, dass der Server zuvor gestartet wurde. Die Tool-direkt Version erfordert keine weiteren Schritte und ist ohne Server lauffähig.

Server.jar sollte wenn möglich über die Eingabeaufforderung(Windows) bzw. das Terminal (Mac) mitttels java -jar Server.jar gestartet werden(Vorher sollte zum Ort des Programmes im Dateisystem navigiert werden), damit die Ausgaben des Servers in der Konsole sichtbar werden. Der Server kann aber auch durch einfaches doppeltes Klicken gestartet werden, dann sind jedoch keine Ausgaben des Servers, inklusive Fehlermeldungen, sichtbar.

Außerdem sollten Server und Client in einem internen gemeinsamen Netz laufen, da die IP-Adresse des Servers im Moment fest im Code des Clients programmiert ist. Die gewünschte IP-Adresse ist den Entwicklern vor der Auslieferung mitzuteilen.

9 Projektbewertung aus Entwicklersicht

Die Entwicklung ist nicht vollständig abgeschlossen, es wäre sinnvoll und wünschenswert die Server-/Client-Anbindung umzustrukturieren und auf die Programmierschnittstelle Jax-Rs umzustellen.

Dies hätte eine höhere Stabilität, höhere Sichheit und geringere Fehlerquote zufolge.