



HSB

Hochschule Bremen
City University of Applied Sciences

Entwicklungsarbeit

Autor: Philipp Hennken

Matrikelnummer: 5264799

E-Mail: phennken@stud.hs-bremen.de

Hochschule Bremen

Fakultät 4 – Elektrotechnik und Informatik

Studiengang: Informatik Software und Systemtechnik dual 1. Semester

Modul 1.5 Einführung in die objektorientierte Programmierung

Prüfer: Prof. Dr. Ing. Heiko Mosemann

Abgabedatum: 27.01.2023

Inhalt

1. Erklärung über das eigenständige Erstellen der Arbeit.....	3
2. Einleitung.....	4
3. Aufgabenstellung	4
3.1. Aufgabenstellung – Teilaufgabe I	4
3.2. Aufgabenstellung – Teilaufgabe II	4
3.3. Aufgabenstellung – Teilaufgabe III	5
3.4. Aufgabenstellung – Teilaufgabe IV	5
3.5. Aufgabenstellung – Teilaufgabe V	6
4. Anforderungsdefinition	6
5. Entwurf.....	6
5.1. Pakete	7
5.2. Aufbau des Programms - Softwareplanung	7
5.2.1. Aufbau des Programms für Teilaufgabe I	8
5.2.2. Aufbau des Programms für Teilaufgabe II	10
5.2.3. Aufbau des Programms für Teilaufgabe III	11
5.2.4. Aufbau des Programms für Teilaufgabe IV	12
5.2.5. Aufbau des Programms für Teilaufgabe V	13
5.2.6. Übersicht der gesamten Software	15
5.3. Zeitmessung	16
6. Benutzungshinweise	16
7. Testdokumentation.....	17
7.1. Test des Aufgabenteil I	17
7.2. Test des Aufgabenteil II	18
7.3. Test des Aufgabenteil III	19
7.4. Test des Aufgabenteil IV	21
7.5. Test des Aufgabenteil V	22
7.6. Test der kompletten Software	23
7.6.1. Test für die Epsilonumgebung	24
7.6.2. Test für die Entfernung zwischen zwei Ladestationen.....	24
7.6.3. Test für die Eingabe der Postleitzahlen.....	25
8. Anwendungsbeispiel	26
9. Verweise.....	27
9.1. Auszüge aus dem Quellcode	27
9.2. Abbildungsverzeichnis:	28
9.3. Onlineverzeichnis	29

9.4.	Tabellenverzeichnis	29
10.	Anhang.....	
10.1.	Quellprogramm	
10.1.1.	Control Package	
10.1.2.	Model Package	
10.1.3.	Res Package	
10.1.4.	Util Package	
10.1.5.	View Package.....	
10.2.	Javadoc Dokumentation	

1. Erklärung über das eigenständige Erstellen der Arbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht.

Diese Erklärung erstreckt sich auch auf in der Arbeit enthaltene Grafiken, Skizzen, bildliche Darstellungen sowie auf Quellen aus dem Internet.

Die Arbeit habe ich in gleicher oder ähnlicher Form auch auszugsweise noch nicht als Bestandteil einer Prüfungs- oder Studienleistung vorgelegt.

Ich versichere, dass die eingereichte elektronische Version der Arbeit vollständig mit der Druckversion übereinstimmt.

Philipp Hennken

Matrikelnummer: 5264799

Wildeshausen, den 26.01.2023

A handwritten signature in black ink, appearing to read 'Hennken', with a stylized, cursive script.

2. Einleitung

Im Rahmen des Moduls „Einführung in die objektorientierte Programmierung (Modul 1.5 PROG) ist die Prüfungsform für das Wintersemester 2022/2023 die Entwicklungsarbeit. Nach Abschnitt II: Prüfungsform § 7 Arten der Prüfungsleistungen, Studienleistungen der Prüfungsordnung umfasst eine Entwicklungsarbeit die Aufgabenstellung, die Anforderungsdefinition, einen Entwurf, das Quellprogramm, die Testdokumentation, Benutzungshinweise sowie ein Anwendungsbeispiel.

3. Aufgabenstellung

Die Aufgabenstellung stammt aus dem Dokument [„Aufgabenstellung Entwicklungsarbeit PROG im WiSe 22/23“](#) (vgl. Link zur Aufgabenstellung für die Entwicklungsarbeit) welches am 21.12.2022 veröffentlicht und vorgestellt wurde.

Die gesamte Aufgabenstellung besteht aus fünf Teilaufgaben, welche chronologisch bearbeitet werden können. Teilaufgabe eins bis drei können auch zusammengefasst bearbeitet werden.

3.1. Aufgabenstellung – Teilaufgabe I

In Teilaufgabe eins wird von der geprüften Person gefordert, dass die Anwendung eine Liste von [esriDeutschland](#) (vgl. Link zu esriDeutschland), welche 21566 Ladesäulen und deren Betreiber, Straße, Hausnummer, Postleitzahl, Ort, Bundesland, Breitengrad, Längengrad und Anschlussleistung beinhaltet, einlesen und in einer gewählten Collection abspeichern soll. Die Daten wurden in Form einer CSV-Datei zum Download bereitgestellt. Die Daten sind zeilenweise gespeichert und die Werte sind getrennt durch Semikolons. Vor der Eintragung in die Collection soll darauf geachtet werden, dass alle Daten plausibel sind, besonders auf die Längen- und Breitengrade soll geachtet werden.

Es sollen sinnvolle Ausgaben für die Einlese- und Bearbeitungszeit gemacht werden.

Eine mögliche sinnvolle Ausgabe ist im Originaldokument¹ gezeigt.

3.2. Aufgabenstellung – Teilaufgabe II

In Teilaufgabe zwei wird von der geprüften Person gefordert, die in Teilaufgabe I erstellte Collection nach Postleitzahlen zu sortieren. Die Ladestation mit der kleinsten Postleitzahl soll dabei ganz oben stehen, die mit der größten ganz unten. Sollten zwei Ladesäulen die gleiche

Postleitzahl haben, soll die Ladesäule mit der höheren Abgabeleistung zuerst aufgelistet sein. Die Bearbeitungszeit für die Sortierung der Collection soll ausgegeben werden.

3.3. Aufgabenstellung – Teilaufgabe III

In Teilaufgabe drei wird von der geprüften Person gefordert, dass der Anwendung zwei Argumente übergeben werden. Zwei natürliche Zahlen, die Epsilonumgebung und die maximale Entfernung zwischen Ladesäulen.

Die Argumente sollen auf Plausibilität überprüft werden. Fehlerhafte Argumente sollen dem Nutzer angezeigt werden.

Sollten die Argumente korrekt sein, sollen die in Teilaufgabe zwei sortierte Liste durchlaufen werden und alle Stationen, die sich innerhalb der Epsilonumgebung der Referenzstation befinden, sollen gelöscht werden.

Die erste Ladestation der Liste wird zur Referenzstation. Die Lage der Referenzstation soll unter Verwendung der Breiten- und Längengrade mit allen anderen verglichen werden. Dazu soll die Haversine-Formel verwendet werden.

Gelöscht wird ein Element, wenn der Abstand zur Referenzstation kleiner oder gleich der Epsilonumgebung ist. Nachdem die ganze Liste durchlaufen wurde, soll die nächste Station der Liste zur Referenzstation werden.

3.4. Aufgabenstellung – Teilaufgabe IV

In Teilaufgabe vier wird von der geprüften Person gefordert, dass die Anwendung einen Graphen implementiert, der ein Wegnetz der Ladestationen abbildet. Jede Ladestation soll einen Knoten abbilden. Ist die Entfernung zweier Knoten kleiner oder gleich der maximalen Entfernung zwischen zwei Ladesäulen, so soll eine Kante diese Knoten verbinden.

Dafür soll ein ADT Graph selbst implementiert werden.

Um die Anzahl der Ladesäulen zu reduzieren und damit die Tests übersichtlich zu halten, kann die Epsilonumgebung angepasst werden.

Für den ADT Graph sollen die impliziten Datenstrukturen Adjazenz-Liste oder Adjazenz-Matrix verwendet werden. Die beiden Möglichkeiten sollen bewertet werden. Anschließend soll sich für die geeignetere Datenstruktur entschieden werden.

Ausgegeben werden soll der Graph sowie die Bearbeitungszeit.

3.5. Aufgabenstellung – Teilaufgabe V

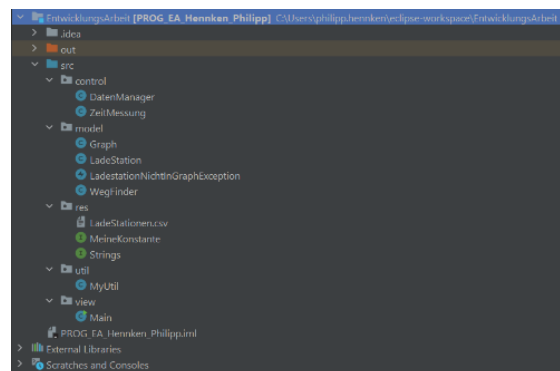
In Teilaufgabe fünf wird von der geprüften Person gefordert, dass die Anwendung überprüft, ob es einen möglichen Weg von einem Start-Knoten zu einem Ziel-Knoten gibt. Der Nutzer soll dafür eine Start-Postleitzahl eingeben können. Diese Eingabe soll auf die Existenz eines Knotenpunktes zu dieser Postleitzahl überprüft werden. Anschließend soll vom Nutzer ein Ziel-Ort angegeben werden. Ebenfalls soll die Bearbeitungszeit des Vorganges angegeben werden.

4. Anforderungsdefinition

Es wird allgemein gefordert, dass objektorientiert programmiert wird. Redundanter Code soll vermieden werden und Methoden sollen modular und überschaubar gehalten werden. Das in dem Modul „Grundlagen der Informatik“ Gelernte soll ebenfalls berücksichtigt werden, dies beinhaltet, dass Methoden nach dem Design-by-Contract-Verfahren mit Vor- und Nachbedingungen kommentiert werden sollen. Kommentare und Dokumentationskommentare sollen sinnvoll sein, Bezeichner sollen sinnvolle und verständliche Namen haben. Literale sollen nur in Ausnahmefällen eingesetzt werden, sonst sollen Konstanten eingesetzt werden. Insgesamt soll der Code einfach und verständlich sein, da sauber Code vor der Performance Vorrang hat.

5. Entwurf

Zur Strukturierung des Programms verwende ich das model-view-control-Konzept. Nach diesem Konzept werden unterschiedliche Teile des Programms in unterschiedliche Pakete verteilt. Neben den Paketen „model“, „view“ und „control“ werden für dieses Programm die Pakete „res“ und „util“ angelegt.



5.1. Pakete

Das „model“ Paket dient der Modellierung und der Festlegung von Eigenschaften sowie dem Steuern des Verhaltens. Es spielt sich die komplette Logik des Programms in diesem Paket ab. Daher befinden sich im Programm die Klassen „Graph“, „LadeStation“, „LadeStationNichtInGraphException“ und „WegFinder“ in diesem Paket.

Das „view“ Paket beinhaltet alles für die Ergebnispräsentation und dient als Schnittstelle zum Benutzer. Deshalb befindet sich im Programm die „Main“ Klasse dort.

Zur Verbindung von dem „view“ Paket und dem „model“ Paket benötigt es das „control“ Paket. Dort findet die Vermittlung sowie die Steuerung statt. Im Programm befinden sich hier die „DatenManager“ sowie die „ZeitMessung“ Klasse.

Das Paket „res“ benutze ich für die Speicherung von benötigten Daten. Dazugehört die CSV-Datei „LadeStationen.csv“, die von dem Programm ausgelesen werden soll sowie die Interfaces „MeineKonstante“ und „Strings“, die Konstanten- und Stringlitterale abspeichern und somit für übersichtlicheren Code sorgen.

Im „util“ Paket befindet sich die „MyUtil“ Klasse. Diese kümmert sich um alle notwendigen Ein- und Ausgaben sowie die verwendete Haversine-Formel, die zur Berechnung von Entfernungen auf einer Kugel eingesetzt wird.

5.2. Aufbau des Programms - Softwareplanung

Das Programm befasst sich insgesamt mit der Verarbeitung von Ladestationen. Daher war mein erster Schritt, eine Instanz der Klasse „LadeStation“ zu modellieren. Eine Ladestation hat mehrere Attribute, die durch die CSV-Datei in der Aufgabenstellung mitgegeben worden sind. Diese wird eine Ladestation benötigen, genauso wie die Getter der Attribute. Setter sind nicht notwendig, da sich der Benutzer nur Ladestationen ausgeben lassen, aber keine eingeben kann.



Abbildung 1: UML- Klassendiagramm einer Ladestation

5.2.1.Aufbau des Programms für Teilaufgabe I

Da in „Aufgabenstellung Teil I“ gesagt ist, dass fehlerhafte Daten bei den Geokoordinaten vorliegen, markiere ich Ladestationen mit falschen Geokoordinaten. Um zu markieren, dass eine Ladestation fehlerhafte Geokoordinaten hat, wird dieser Ladestation nur der Ort und der Betreiber zugewiesen. (Auszug aus dem Quellcode – 1, Zeilen 23 und 24) Allen anderen Attributen wird „null“ zugewiesen. Später werden Ladestationen, die den Wert „null“, beispielsweise als Straße haben, gelöscht. (Auszug aus dem Quellcode – 2, Zeilen 18 bis 24) Allen Ladestationen mit korrekten Geokoordinaten werden alle Attribute zugewiesen. (Auszug aus dem Quellcode – 1, Zeilen 12 bis 20)

```
1  public LadeStation (String input)
2  {
3      String[] splits = input.split(Strings.SIMIKOLON);
4      if(
5          Float.parseFloat(splits[MeineKonstante.BREITENGRAD_INDEX]) > 0 &&
6          Float.parseFloat(splits[MeineKonstante.BREITENGRAD_INDEX]) < 90 &&
7          Float.parseFloat(splits[MeineKonstante.LAENGENGRAD_INDEX]) > 0 &&
8          Float.parseFloat(splits[MeineKonstante.LAENGENGRAD_INDEX]) < 180
9      )
10     {
11         this.betreiber = splits[MeineKonstante.BETREIBER_INDEX];
12         this.strasse = splits[MeineKonstante.STRASSE_INDEX];
13         this.hausnummer = splits[MeineKonstante.HAUSNUMMER_INDEX];
14         this.postleitzahl = Integer.parseInt(splits[MeineKonstante.POSTLEITZAHL_INDEX]);
15         this.ort = splits[MeineKonstante.ORT_INDEX];
16         this.bundesland = splits[MeineKonstante.BUNDESLAND_INDEX];
17         this.breitengrad = Float.parseFloat(splits[MeineKonstante.BREITENGRAD_INDEX]);
18         this.laengengrad = Float.parseFloat(splits[MeineKonstante.LAENGENGRAD_INDEX]);
19         this.anschlussleistung =
20             Float.parseFloat(splits[MeineKonstante.ANSCHLUSSLEISTUNG_INDEX]);
21     }else
22     {
23         this.betreiber = splits[MeineKonstante.BETREIBER_INDEX];
24         this.ort = splits[MeineKonstante.ORT_INDEX];
25     }
26 }
```

Auszug aus dem Quellcode - 1

Fehlerhafte Geokoordinaten liegen dann vor, wenn sie außerhalb des Wertebereichs der Breiten- beziehungsweise Längengrade liegen. Der Wertebereich für den Breitengrad ist 90°S bis 90°N und der Wertebereich für die Längengrade sind 180°W bis 180°O. (vgl. Link zum Wikipedia Artikel über Geographische Koordinaten) Da Deutschland komplett östlich vom Nullmeridian und nördlich vom Äquator liegt, können die Wertebereiche 0 bis 90 beziehungsweise 0 bis 180 angegeben werden. (Auszug aus dem Quellcode – 1, Zeilen 5 bis 8)

Diese Ladestationen müssen dann in einer Collection abgespeichert werden, um mit ihnen arbeiten zu können. Daher speichert das Programm alle eingelesenen Ladestationen in einer ArrayList ab. (Auszug aus dem Quellcode – 2, Zeilen 8 bis 12) Die ArrayList habe ich gewählt, da ich mit dieser Collection bereits gearbeitet habe und daher wusste, dass die ArrayList alles für diese Aufgabe Notwendige mitbringt. Dies geschieht im ersten Teil der Methode „leseDatenZeilenweise“ in der Klasse „DatenManager“.

Im zweiten Teil dieser Methode werden die Ladestationen mit fehlerhaften Geokoordinaten herausgelöscht. (Auszug aus dem Quellcode – 2, Zeilen 18 bis 24)

```
1  public ArrayList<LadeStation> leseDatenZeilenweise(BufferedReader reader)
2  {
3      ZeitMessung.start();
4      String line = null;
5      ArrayList<LadeStation> arrayList = new ArrayList<>();
6      try
7      {
8          while ((line = reader.readLine()) != null)
9          {
10             arrayList.add(new LadeStation(line));
11             anzahlEitraege++;
12         }
13     }
14     catch (IOException e)
15     {
16     }
17     for (int i = 0; i < arrayList.size(); i++)
18     {
19         if (arrayList.get(i).getStrasse() == null)
20         {
21             MyUtil.normalAusgeben(Strings.LADESTATION_PRE_ANBIETER +
22                 arrayList.get(i).getBetreiber() + Strings.LADESTATION_POST_ANBIETER);
23             arrayList.remove(i);
24             i--;
25             anzahlEitraege--;
26         }
27     }
28     MyUtil.normalAusgeben(Strings.ES_SIND + getAnzahlEitraege() +
29         Strings.LADESTATIONEN_EINGETRAGEN);
30     MyUtil.normalAusgeben(Strings.EINLESEN_ZEIT +
31         ZeitMessung.berechneVergangeneZeitInMillisekunden() + Strings.EINHEIT_ZEITMESSUNG);
32     return arrayList;
33 }
```

Auszug aus dem Quellcode - 2

5.2.2.Aufbau des Programms für Teilaufgabe II

Für Aufgabenteil II verwende ich zum Sortieren der Liste die „sort“-Methode der ArrayList. Dafür schreibe ich einen eigenen Comperator der zuerst aufsteigend nach Postleitzahlen und als zweites Argument absteigend nach der Anschlussleistung sortiert. Nach der Anschlussleistung wird nur sortiert, wenn die Postleitzahlen gleich sind.

```
1 public ArrayList<LadeStation> sortieren (ArrayList<LadeStation> arrayListZuSortieren)
2 {
3     ZeitMessung.start();
4     arrayListZuSortieren.sort((LadeStation o1, LadeStation o2) ->
5     {
6         if (o1.getPostleitzahl() == o2.getPostleitzahl())
7         {
8             return o2.getAnschlussleistung().compareTo(o1.getAnschlussleistung());
9         } else
10        {
11            return Integer.valueOf(o1.getPostleitzahl()).compareTo(o2.getPostleitzahl());
12        }
13    });
14    MyUtil.normalAusgeben(Strings.AUSGABE_LISTE_WURDE_SORTIERT);
15    MyUtil.normalAusgeben(Strings.SORTIEREN_ZEIT +
16        ZeitMessung.berechneVergangeneZeitInMillisekunden() + Strings.EINHEIT_ZEITMESSUNG);
17    return arrayListZuSortieren;
18 }
```

Auszug aus dem Quellcode - 3

5.2.3.Aufbau des Programms für Teilaufgabe III

In Aufgabenteil III, dem Aussortieren von zu nah aneinander liegenden Ladesäulen, benutze ich die Methode „loescheEintraegeInEpsilonUmgebung“ in der „DatenManager“ Klasse. Auszug aus dem Quellcode – 4) Diese nutzt zum ersten Mal die Haversine-Formel, die zur Berechnung von Entfernungen einzusetzen ist, auch wenn man dafür davon ausgehen muss, dass die Welt eine exakte Kugel ist. (Auszug aus dem Quellcode – 4, Zeile 8)

Die Methode funktioniert so, dass die erste Ladestation der Liste als Referenzstation gesetzt wird und anschließend jedes Element hinter der Referenzstation einzeln und nacheinander steht als Vergleichsstation gesetzt wird. (Auszug aus dem Quellcode – 4, Zeilen 4 bis 14) Dann wird die Entfernung zwischen der Referenzstation und der Vergleichsstation mittels der Haversine-Formel berechnet. (Auszug aus dem Quellcode – 4, Zeile 8) Ist diese Entfernung kleiner als die Epsilonumgebung, wird die Vergleichsstation gelöscht und die nächste Station ist die Vergleichsstation. (Auszug aus dem Quellcode – 4, Zeilen 10 und 11) Nachdem die Referenzstation mit allen Nachfolgenden verglichen wurde, wird die nächste, noch übrige Ladestation zur Referenzstation. Dieser Vorgang wird wiederholt, bis die letzte Ladestation der Liste zur Referenzstation wurde. (Auszug aus dem Quellcode – 4, Zeile 6) Am Ende wird die Zeit für die Bearbeitungszeit des Sortierens ausgegeben und die sortierte Liste wird zurückgegeben. (Auszug aus dem Quellcode – 4, Zeilen 15 und 16)

```
1 public ArrayList<LadeStation> loescheEintraegeInEpsilonUmgebung(ArrayList<LadeStation>
  sortierteListe, int epsilonUmgebung)
2 {
3     ZeitMessung.start();
4     for (int i = 0; i < sortierteListe.size(); i++)
5     {
6         for (int j = i+1; j < sortierteListe.size(); j++)
7         {
8             if(MyUtil.haversine(sortierteListe.get(i).getBreitengrad(),
              sortierteListe.get(i).getLaengengrad(), sortierteListe.get(j).getBreitengrad(),
              sortierteListe.get(j).getLaengengrad()) < epsilonUmgebung)
9             {
10                 sortierteListe.remove(j);
11                 j--;
12             }
13         }
14     }
15     MyUtil.normalAusgeben(Strings.LOESCHEN_ZEIT +
      ZeitMessung.berechneVergangeneZeitInMillisekunden() + Strings.EINHEIT_ZEITMESSUNG);
16     return sortierteListe;
17 }
```

Auszug aus dem Quellcode - 4

5.2.4. Aufbau des Programms für Teilaufgabe IV

Für Teilaufgabe IV soll ein Graph implementiert werden. Zur Implementierung des Graphen wähle ich eine Adjazenz-Liste. Eine andere Möglichkeit wäre die Adjazenz-Matrix gewesen. Ich habe mich bewusst für die Adjazenz-Liste entschieden, da sie weniger Speicherplatz benötigt und für den simplen Suchalgorithmus, den ich verwende, ausreicht. Der Unterschied liegt darin, dass eine Matrix entweder „true“ oder „false“ als Verbindung zweier Ladestationen eintragen würde und die Liste nur die „wahren“ Verbindungen. So können alle „false“-Werte, die ich zu keinem Zeitpunkt im Programm benötige, vernachlässigt werden und benötigen keinen Speicherplatz.

Dazu verwende ich eine HashMap mit einem Objekt der Klasse „LadeStation“ als Key und einer ArrayList bestehend aus Objekten der Klasse „LadeStation“ als Value. (Auszug aus dem Quellcode – 5, Zeile 1) Jeder Key ist dabei ein Knoten. Eine Kante ist dann vorhanden, wenn sich eine Ladestation in der ArrayList befindet, die als Value der Ladestation zugeordnet ist, die der Key ist. Erstellt wird der Graph in der Klasse „Graph“ mit der Methode „erzeugeGraphen“. (Auszug aus dem Quellcode – 5) Dieser wird nur die Reichweite des Autos übergeben. Im Konstruktor wurde vorher die ArrayList mit allen sortierten Ladestationen übergeben, sodass die Methode auch darauf zugreifen kann. Jedes Element der Liste wird durch die erste Schleife zu einem Knoten gemacht. (Auszug aus dem Quellcode – 5, Zeile 5) In jedem

Knoten wird erneut durch alle Ladestationen der Liste iteriert und überprüft, dass die Entfernung zwischen den Stationen kleiner ist als die Reichweite des Autos. Dann kommt die Ladestation in die ArrayList und gleichzeitig ist damit gespeichert, dass eine Kante zwischen den Ladestationen besteht. (Auszug aus dem Quellcode – 5, Zeilen 15 bis 18) Damit nicht die gleiche Station als Key und als Value abgespeichert wird, wird vorher überprüft, dass es nicht die gleiche Station ist. (Auszug aus dem Quellcode – 5, Zeilen 10 bis 13)

```
1 public HashMap<LadeStation, ArrayList<LadeStation>> erzeugeGarphen (int reichweiteAuto)
2 {
3     ZeitMessung.start();
4     HashMap<LadeStation, ArrayList<LadeStation>> graph = new HashMap<>();
5     for (int i = 0; i < arrayListMitLadeStationen.size(); i++)
6     {
7         ArrayList<LadeStation> vorruebergendeArrayList = new ArrayList<>();
8         for (int j = 0; j < arrayListMitLadeStationen.size(); j++)
9         {
10             if(j==i)
11             {
12                 continue;
13             }
14
15             if (reichweiteAuto > MyUtil.haversine(arrayListMitLadeStationen.get(i).getBreitengrad(),
16                 arrayListMitLadeStationen.get(i).getLaengengrad(),
17                 arrayListMitLadeStationen.get(j).getBreitengrad(),
18                 arrayListMitLadeStationen.get(j).getLaengengrad()))
19             {
20                 vorruebergendeArrayList.add(arrayListMitLadeStationen.get(j));
21             }
22             graph.put(arrayListMitLadeStationen.get(i), vorruebergendeArrayList);
23         }
24         this.setKnoten(graph);
25         MyUtil.normalAusgeben(Strings.GRAPH_ERSTELLEN +
26             ZeitMessung.berechneVergangeneZeitInMillisekunden() + Strings.EINHEIT_ZEITMESSUNG);
27     }
28     return graph;
29 }
```

Auszug aus dem Quellcode - 5

5.2.5.Aufbau des Programms für Teilaufgabe V

Die „WegFinder“-Klasse dient der Teilaufgabe V. Zum Finden des Wegs benötigt es erstmal eine Methode, um von der eingegebenen Postleitzahl zu einer dazugehörigen Ladestation zu gelangen. (Auszug aus dem Quellcode – 6) Dafür implementiere ich die „erhalteLadeStationVonPostleitzahl“-Methode. Dieser Methode wird eine Postleitzahl sowie ein Graph übergeben. (Auszug aus dem Quellcode – 6, Zeile 1) Dann wird durch alle Keys des

Graphen iteriert und geschaut, ob eine Postleitzahl einer Ladestation des Graphen gleich der eingegebenen Postleitzahl ist. In dem Fall wird die gesuchte Ladestation zurückgegeben. (Auszug aus dem Quellcode – 6, Zeilen 3 bis 8) Sollte eine solche Ladestation nicht vorhanden sein wird „null“ zurückgegeben, sodass eine NullPointerException gefangen werden kann. (Auszug aus dem Quellcode – 6, Zeile 10) Durch das Abfangen dieser Exception wird dann der Benutzer dazu aufgefordert, eine andere Postleitzahl als Start- beziehungsweise Endpostleitzahl einzugeben.

```
1  public static Ladestation erhalteLadestationVonPostleitzahl (int postleitzahl, Graph graph)
2  {
3      for (Ladestation ladestation : graph.getGraphKeys())
4      {
5          if (ladestation.getPostleitzahl() == postleitzahl)
6          {
7              return ladestation;
8          }
9      }
10     return null;
11 }
```

Auszug aus dem Quellcode - 6

Die eigentliche Suche nach einem Weg findet in der „findeWeg“-Methode statt. (Auszug aus dem Quellcode – 7) Zuerst wird die Startladestation der Ladestation zugewiesen, dessen zugehörige Postleitzahl als Startpostleitzahl eingegeben wurde. (Auszug aus dem Quellcode – 7, Zeile 8) Es werden anschließend alle zu der Ladestation verbundenen Ladestationen durchsucht. (Auszug aus dem Quellcode – 7, Zeile 9) Wenn Verbindungen existieren, werden alle Ladestationen, die als Value in der ArrayList des Graphen gespeichert sind, durchsucht. Sollte schon eine Ladestation dabei sein, die die gleiche Postleitzahl hat wie die eingegebene Zielpostleitzahl, so wird das Programm nach der Ausgabe der Existenz eines Weges beendet. (Auszug aus dem Quellcode – 7, Zeilen 13 bis 16) Wenn noch keine passende Ladestation dabei ist, wird die überprüfte Ladestation der ArrayListe „besuchteLadestationen“ hinzugefügt, damit verhindert wird, dass Ladestationen doppelt untersucht werden und es nicht zu einer Endlosschleife kommt. (Auszug aus dem Quellcode – 7, Zeilen 11 und 18)

```

1      public void findeWeg(Graph graph, int startPLZ, int endPLZ, ArrayList<LadeStation>
      besuchteLadeStationen) throws LadestationNichtInGraphException
2      {
3          ZeitMessung.start();
4          this.startPLZ = startPLZ;
5          this.endPLZ = endPLZ;
6          this.graph = graph;
7          testeObStartUndZielPostleitzahlImGraphenSind();
8          Ladestation startLadeStation = DataManager.erhalteLadeStationVonPostleitzahl(startPLZ, graph);
9          for (Ladestation ladeStation:graph.getGraph().get(startLadeStation))
10         {
11             if (! besuchteLadeStationen.contains(ladeStation))
12             {
13                 if (ladeStation.getPostleitzahl() == endPLZ)
14                 {
15                     wegIstVorhanden();
16                 }else
17                 {
18                     besuchteLadeStationen.add(startLadeStation);
19                     findeWeg(graph, ladeStation.getPostleitzahl(), endPLZ, besuchteLadeStationen);
20                 }
21             }
22         }
23         keinWegVorhanden();
    
```

Auszug aus dem Quellcode - 7

5.2.6.Übersicht der gesamten Software

Das gesamte Programm lässt sich übersichtlich als UML-Diagramm darstellen.

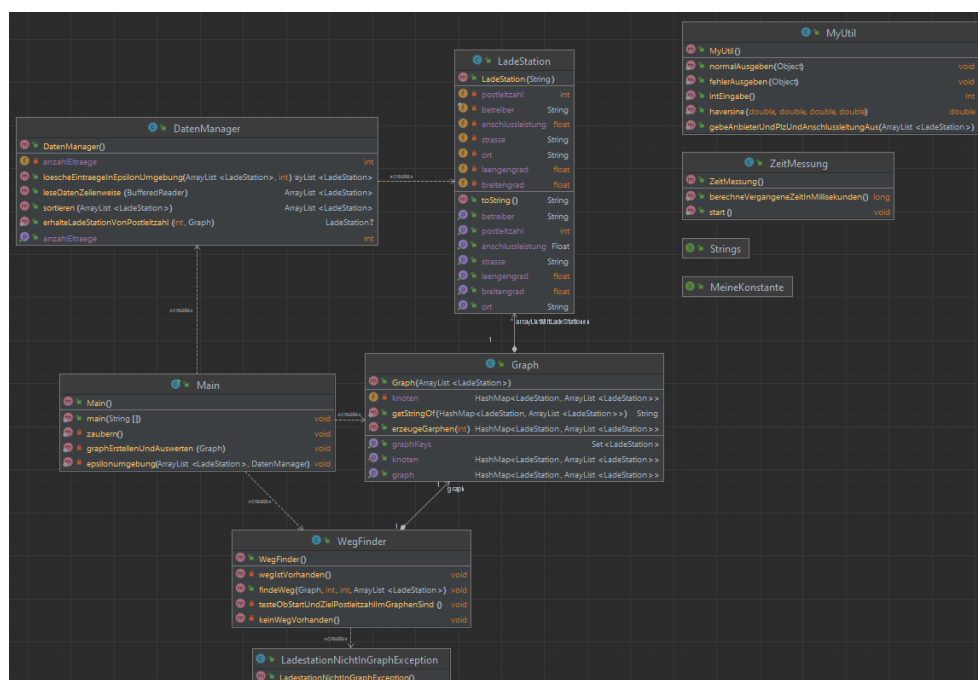


Abbildung 2: UML-Diagramm der Software

5.3. Zeitmessung

Bei jedem Aufgabenteil wurde gefordert, die Bearbeitungsdauer anzugeben. Um einen Mittelwert zu bilden, der die durchschnittlich benötigte Zeit angibt, führe ich das Programm fünf Mal aus. Die eingegebenen Werte werden dabei nicht verändert. Die Epsilonumgebung wird auf 25 und die maximale Entfernung zwischen zwei Ladesäulen wird auf 40 gesetzt. Anschließend wird ein Weg von 87435 München nach 54291 Trier gesucht. Dabei sind die Zeiten in Millisekunden angegeben und nach den Teilaufgaben gegliedert.

Teilaufgabe	I	II	III	IV	V	Gesamtzeit
Test 1	160	9	329	20	3	521
Test 2	144	8	369	20	3	544
Test 3	153	9	333	21	2	518
Test 4	151	10	323	20	4	508
Test 5	154	9	341	19	3	526
Durchschnitt:	152,4	9	339	20	3	523,4

Tabelle 1: Tabelle mit gemessenen Zeiten

6. Benutzungshinweise

Das Programm ist sehr benutzerfreundlich geschrieben, sodass falsche Eingaben einfach korrigiert werden können. Für einen reibungslosen Ablauf ohne Fehlermeldungen sollten alle Eingaben natürliche Zahlenwerte sein.

Für die Epsilonumgebung sollten diese Zahlen maximal 587 sein, sonst ist nur eine Ladestation in der Liste übrig.

Die maximale Entfernung zwischen zwei Ladesäulen sollte größer sein als die Epsilonumgebung. Ansonsten wird der Graph keine Verbindung zwischen Ladesäulen finden. Somit kann auch nie ein Weg gefunden werden.

Die Start- und Zielpostleitzahlen sollten natürliche Zahlen zwischen 1000 und 99999 sein, die im angezeigten Graphen angegeben sind. Andernfalls kann man die Eingabe korrigieren. Bei einer führenden 0, kann diese ignoriert werden, muss sie aber nicht.

7. Testdokumentation

Um die einzelnen Methoden zu testen, werde ich alle Methoden einzeln aufrufen und mit unterschiedlichen Eingaben und Überprüfungen testen, ob die Methoden das Geforderte erfüllen.

7.1. Test des Aufgabenteil I

Um das korrekte Einlesen und das Aussortieren fehlerhafter Datensätze zu testen, rufe ich in der Main-Methode die Methode „test1“ auf. Diese ruft die „leseDatenZeilenweise“-Methode auf und soll die erstellte ArrayList ausgeben. (Auszug aus dem Quellcode – 8, Zeile 4) Zur Ausgabe verwende ich die „gebeAnbieterUndPlzUndAnschlussleitungAus“-Methode, die von einer Ladestation nur den Anbieter, die Postleitzahl und die Anschlussleistung ausgibt. (Auszug aus dem Quellcode – 8, Zeile 5)

```
1 private static void test1() throws FileNotFoundException
2 {
3     DatenManager datenManager = new DatenManager();
4     ArrayList<LadeStation> ladeStations = datenManager.leseDatenZeilenweise(new BufferedReader(new
        FileReader(Strings.FILE_NAME)));
5     MyUtil.gebeAnbieterUndPlzUndAnschlussleitungAus(ladeStations);
6 }
```

Auszug aus dem Quellcode - 8

Im Datensatz habe ich in Zeile 18389 eine Ladestation gefunden, die fehlerhafte Geokoordinaten besitzt. Damit kann ich später überprüfen, ob diese Ladestation noch in der Liste ist.

```
18386 Lidl Dienstleistung GmbH & Co. KG;Kurfürstenstr.;69;54516;Wittlich;Rheinland-Pfalz;49.98304
18387 Autohaus Kainz GmbH & Co. KG;Max-Planck-Str.;21;54516;Wittlich;Rheinland-Pfalz;49.988035;6.
18388 Autohaus Scheider GmbH;Kurfürstenstraße;41;54516;Wittlich;Rheinland-Pfalz;49.984689;6.89542
18389 W. Kröfges GmbH & CO KG;Moseltalstr;40;54518;Osann-Monzel;Rheinland-Pfalz;49.919;6.952;72
18390 EnBW mobility+ AG und Co.KG ;Autobahnraststätte Eifel West;0;54533;Niederöfflingen;Rheinlan
18391 EnBW mobility+ AG und Co.KG ;Raststätte Eifel Ost;0;54533;Niederöfflingen;Rheinland-Pfalz;5
18392 EnBW mobility+ AG und Co.KG ;Autobahnraststätte Eifel Ost;0;54533;Niederöfflingen;Rheinland-
18393 EnBW mobility+ AG und Co.KG ;Autobahn;0;54533;Niederöfflingen;Rheinland-Pfalz;50.0684783;6.8
```

Abbildung 3: Auszug aus der CSV-Datei

Wie in der Ausgabe über die Konsole zu sehen, ist die Ladestation nicht mehr in der Liste. Es wurde dem Nutzer ausgegeben, dass die Ladestation nicht in die Liste eingetragen wurde, ebenfalls sieht man, dass an der Stelle 18.389 in der Liste nun eine andere Ladesäule steht.

```
Eine Ladestation von 'W. Kröfges GmbH & CO KG' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
```

Abbildung 4: Ausgabe einer nicht eingetragenen Ladestation

```
Run: Main
18387: innogy SE; Postleitzahl: 56253; Anschlussleistung: 44.0 Kilowatt
18388: innogy eMobility Solutions GmbH; Postleitzahl: 56759; Anschlussleistung: 44.0 Kilowatt
18389: innogy SE; Postleitzahl: 56766; Anschlussleistung: 44.0 Kilowatt
18390: Energieversorgung Mittelrhein AG; Postleitzahl: 56812; Anschlussleistung: 22.0 Kilowatt
18391: AHG-Newel GmbH; Postleitzahl: 56812; Anschlussleistung: 39.6 Kilowatt
18392: innogy SE; Postleitzahl: 56814; Anschlussleistung: 44.0 Kilowatt
```

Abbildung 5: Ladestation an der Stelle 18.389

Die Methode gibt am Ende Informationen über die Zeit, die es benötigte, um alle Daten auszulesen und fehlerhafte Daten auszusortieren. Ebenfalls wird angezeigt, dass die Liste 25.000 fehlerfreie Ladestationen abgespeichert hat.

```
Run: Main
Eine Ladestation von 'Eider-Treene-Sorge GmbH' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'innogy SE' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'innogy SE' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'innogy SE' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'innogy SE' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'innogy SE' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'Stadtwerke Eckernförde GmbH' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'Stadtwerke Nortorf AG' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'Eider-Treene-Sorge GmbH' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'Pattburg Poetzsch GmbH & Co. KG' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'Eisenacher Versorgungs-Betriebe GmbH' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'Thüringer Energie AG' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'Thüringer Energie AG' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'Thüringer Energie AG' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'Energieversorgung Gera GmbH' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'Thüringer Energie AG' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'Comfortharge GmbH' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Eine Ladestation von 'Stadtwerke Muhlhausen GmbH' wurde nicht hinzugefügt, da die Geokoordinaten fehlerhaft sind.
Es sind 21500 Ladestationen mit validen Argumenten in der Liste.
Das Einlesen und Überprüfen der Daten dauerte: 251 Millisekunden

Process finished with exit code 0
```

Abbildung 6: Konsolenausgabe Teilaufgabe I

7.2. Test des Aufgabenteil II

Um zu testen, ob die Sortierung der Ladestationen nach den gegebenen Kriterien funktioniert hat, wird die gesamte Liste der Ladestationen einmal vor und nach der Sortierung aufgerufen. (Auszug aus dem Quellcode – 9, Zeilen 5 bis 7) Der Vergleich wird dann zeigen, ob die Liste sortiert ist. Um nicht 25.000 Daten beim Testen verwenden zu müssen, habe ich eine zweite CSV-Datei erstellt, die aus 100 Ladesäulen der originalen CSV-Datei stammen.

```
1 private static void test2() throws FileNotFoundException
2 {
3     DataManager datenManager = new DataManager();
4     ArrayList<LadeStation> ladeStations = datenManager.leseDatenZeilenweise(new BufferedReader(new
        FileReader(Strings.FILE_NAME_ZUM_TESTEN)));
5     MyUtil.gebeAnbieterUndPlzUndAnschlussleistungAus(ladeStations);
6     datenManager.sortieren(ladeStations);
7     MyUtil.gebeAnbieterUndPlzUndAnschlussleistungAus(ladeStations);
8 }
```

Auszug aus dem Quellcode - 9

Es werden die 100 Ladesäulen eingelesen und ausgegeben. Anschließend werden sie sortiert und erneut ausgegeben. In der Konsole kann man dann die Ausgaben vergleichen.

Die Ausgabe zeigt folgendes.

```
82: EnBW mobility+ AG und Co.KG ; Postleitzahl: 71277; Anschlussleistung: 300.0 Kilowatt
83: deer GmbH; Postleitzahl: 71287; Anschlussleistung: 44.0 Kilowatt
84: EnBW mobility+ AG und Co.KG ; Postleitzahl: 71287; Anschlussleistung: 30.0 Kilowatt
85: EnBW mobility+ AG und Co.KG ; Postleitzahl: 71287; Anschlussleistung: 30.0 Kilowatt
86: Stadtwerk am See GmbH & Co. KG; Postleitzahl: 88045; Anschlussleistung: 22.0 Kilowatt
87: Stadtwerk am See GmbH & Co. KG; Postleitzahl: 88662; Anschlussleistung: 40.0 Kilowatt
88: Stadtwerk am See GmbH & Co. KG; Postleitzahl: 88662; Anschlussleistung: 44.0 Kilowatt
89: Stadtwerk am See GmbH & Co. KG; Postleitzahl: 88662; Anschlussleistung: 44.0 Kilowatt
90: Stadtwerk am See GmbH & Co. KG; Postleitzahl: 88662; Anschlussleistung: 44.0 Kilowatt
91: Stadtwerk am See GmbH & Co. KG; Postleitzahl: 88662; Anschlussleistung: 44.0 Kilowatt
92: Stadtwerk am See GmbH & Co. KG; Postleitzahl: 88662; Anschlussleistung: 44.0 Kilowatt
93: Stadtwerk am See GmbH & Co. KG; Postleitzahl: 88662; Anschlussleistung: 44.0 Kilowatt
94: EnBW mobility+ AG und Co.KG ; Postleitzahl: 88682; Anschlussleistung: 30.0 Kilowatt
95: EnBW mobility+ AG und Co.KG ; Postleitzahl: 88682; Anschlussleistung: 30.0 Kilowatt
96: EDEKA Breisach; Postleitzahl: 79206; Anschlussleistung: 44.0 Kilowatt
97: Energiedienst Holding AG; Postleitzahl: 79224; Anschlussleistung: 22.0 Kilowatt
98: Energiedienst Holding AG; Postleitzahl: 79227; Anschlussleistung: 40.0 Kilowatt
99: Energiedienst Holding AG; Postleitzahl: 79227; Anschlussleistung: 44.0 Kilowatt
100: Energiedienst Holding AG; Postleitzahl: 79227; Anschlussleistung: 44.0 Kilowatt
```

Abbildung 7: Ausgabe der unsortierten Liste

```
82: SWU Energie GmbH; Postleitzahl: 89143; Anschlussleistung: 22.0 Kilowatt
83: SWU Energie GmbH; Postleitzahl: 89143; Anschlussleistung: 22.0 Kilowatt
84: SWU Energie GmbH; Postleitzahl: 89143; Anschlussleistung: 22.0 Kilowatt
85: Lidl Dienstleistung GmbH & Co. KG; Postleitzahl: 89160; Anschlussleistung: 75.0 Kilowatt
86: EnBW mobility+ AG und Co.KG ; Postleitzahl: 89160; Anschlussleistung: 30.0 Kilowatt
87: Gemeinde Dornstadt; Postleitzahl: 89160; Anschlussleistung: 22.0 Kilowatt
88: Gemeinde Dornstadt; Postleitzahl: 89160; Anschlussleistung: 22.0 Kilowatt
89: Gemeinde Dornstadt; Postleitzahl: 89160; Anschlussleistung: 20.0 Kilowatt
90: Gemeinde Dornstadt; Postleitzahl: 89160; Anschlussleistung: 20.0 Kilowatt
91: Donau-Iller Bank eG; Postleitzahl: 89195; Anschlussleistung: 93.0 Kilowatt
92: SWU Energie GmbH; Postleitzahl: 89195; Anschlussleistung: 22.0 Kilowatt
93: SWU Energie GmbH; Postleitzahl: 89195; Anschlussleistung: 22.0 Kilowatt
94: EnBW Ostwürttemberg DonauRies AG; Postleitzahl: 89197; Anschlussleistung: 22.0 Kilowatt
95: SWU Energie GmbH; Postleitzahl: 89198; Anschlussleistung: 44.0 Kilowatt
96: EnBW mobility+ AG und Co.KG ; Postleitzahl: 89579; Anschlussleistung: 30.0 Kilowatt
97: EnBW mobility+ AG und Co.KG ; Postleitzahl: 89584; Anschlussleistung: 98.0 Kilowatt
98: EnBW mobility+ AG und Co.KG ; Postleitzahl: 89584; Anschlussleistung: 30.0 Kilowatt
99: EnBW mobility+ AG und Co.KG ; Postleitzahl: 89584; Anschlussleistung: 30.0 Kilowatt
100: EnBW mobility+ AG und Co.KG ; Postleitzahl: 89584; Anschlussleistung: 22.0 Kilowatt
```

Abbildung 8: Ausgabe der sortierten Liste

Man kann erkennen, dass bei der sortierten Liste die Postleitzahlen alle aufsteigend sind, bei gleicher Postleitzahl sind die Anschlussleistungen absteigend angeordnet. Das ist in der unsortierten Liste nicht der Fall.

7.3. Test des Aufgabenteil III

In Aufgabenteil III sollen Ladestationen gelöscht werden, die zu nah aneinander sind. Wenn die Methode aufgerufen wird, werden je nach Wert der Epsilonumgebung unterschiedlich viele Ladestationen wegfallen. (Auszug aus dem Quellcode – 10, Zeile 6) Da die Haversine-Formel durch die Aufgabenstellung gegeben war, teste ich diese nicht aus und gehe von einer korrekten Funktion aus. Es gilt, dass die eingegebene Epsilonumgebung eine ganze Zahl sein muss.

Das Programm wartet nach dem Einlesen und sortieren der Daten auf eine Eingabe.

```
1 private static void test3() throws IOException
2 {
3     DatenManager datenManager = new DatenManager();
4     ArrayList<LadeStation> ladeStations = datenManager leseDatenZeilenweise(new BufferedReader(new
        FileReader(Strings.FILE_NAME_ZUM_TESTEN)));
5     datenManager.sortieren(ladeStations);
6     datenManager.loescheEintraegeInEpsilonUmgebung(ladeStations, MyUtil.intEingabe());
7     MyUtil.gebeAnbieterUndPlzUndAnschlussleistungAus(ladeStations);
8 }
```

Auszug aus dem Quellcode - 10

```
"C:\Program Files\Java\jdk-18.0.2\bin\java.exe" -Dvisualvm.id=711470388084000 "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.2\lib\idea_rt.jar=63893:C:\Program
Files\JetBrains\IntelliJ IDEA 2022.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\philipp
.hennken\workspace\EntwicklungsArbeit\out\production\PROG_EA_Hennken_Philipp view.Main
Es sind 100 Ladestationen mit validen Argumenten in der Liste.
Das Einlesen und Überprüfen der Daten dauerte: 12 Millisekunden
Die Liste wurde nach Postleitzahlen aufsteigend sortiert. Bei gleicher Postleitzahl ist die Ladestation mit der höheren Leistung zuerst in der Liste.
Das Sortieren der Ladestationen dauerte: 1 Millisekunden
|
```

Abbildung 9: Das Programm fordert eine Eingabe

Dort können unterschiedliche Werte für die Epsilonumgebung eingetragen werden. Gibt man „0“ ein, werden alle 100 Ladesäulen behalten. Gibt man beispielsweise „30“ als Epsilonumgebung ein, bleiben nur 5 Ladestationen übrig.

```
Es sind 100 Ladestationen mit validen Argumenten in der Liste.
Das Einlesen und Überprüfen der Daten dauerte: 12 Millisekunden
Die Liste wurde nach Postleitzahlen aufsteigend sortiert. Bei gleicher Postleitzahl ist die Ladestation mit der höheren Leistung zuerst in der Liste.
Das Sortieren der Ladestationen dauerte: 1 Millisekunden
30
Das Löschen redundanter Daten dauerte: 0 Millisekunden
1: EnBW mobility+ AG und Co.KG ; Postleitzahl: 71065; Anschlussleistung: 315.0 Kilowatt
2: Alwerk GmbH & Co. KG; Postleitzahl: 72535; Anschlussleistung: 22.0 Kilowatt
3: EDEKA Breisach; Postleitzahl: 79206; Anschlussleistung: 44.0 Kilowatt
4: Stadtwerk am See GmbH & Co. KG; Postleitzahl: 88045; Anschlussleistung: 22.0 Kilowatt
5: EnBW mobility+ AG und Co.KG ; Postleitzahl: 88400; Anschlussleistung: 73.0 Kilowatt
Process finished with exit code 0
|
```

Abbildung 10: Löschen mit einer Epsilonumgebung von 30

Bei einer Eingabe von „100“ bleiben nur 3 Ladestation übrig.

```
"C:\Program Files\Java\jdk-18.0.2\bin\java.exe" -Dvisualvm.id=711617181990900 "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.2\lib\idea_rt.jar=63935:C:\Program
Files\JetBrains\IntelliJ IDEA 2022.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\philipp
.hennken\workspace\EntwicklungsArbeit\out\production\PROG_EA_Hennken_Philipp view.Main
Es sind 100 Ladestationen mit validen Argumenten in der Liste.
Das Einlesen und Überprüfen der Daten dauerte: 12 Millisekunden
Die Liste wurde nach Postleitzahlen aufsteigend sortiert. Bei gleicher Postleitzahl ist die Ladestation mit der höheren Leistung zuerst in der Liste.
Das Sortieren der Ladestationen dauerte: 1 Millisekunden
100
Das Löschen redundanter Daten dauerte: 0 Millisekunden
1: EnBW mobility+ AG und Co.KG ; Postleitzahl: 71065; Anschlussleistung: 315.0 Kilowatt
2: EDEKA Breisach; Postleitzahl: 79206; Anschlussleistung: 44.0 Kilowatt
3: Stadtwerk am See GmbH & Co. KG; Postleitzahl: 88045; Anschlussleistung: 22.0 Kilowatt
Process finished with exit code 0
|
```

Abbildung 11: Löschen mit einer Epsilonumgebung von 100

7.4. Test des Aufgabenteil IV

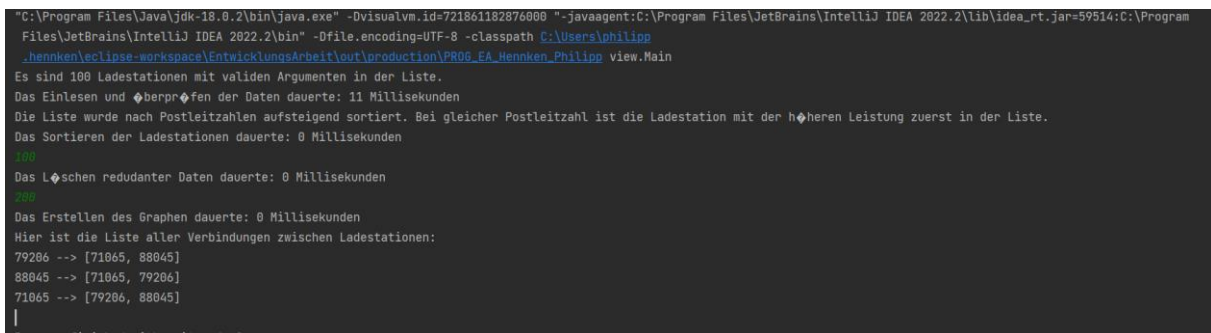
Um zu schauen, ob ein Graph korrekt erzeugt wird, erzeuge ich einen Graphen mit wenigen Ladestationen. Somit kann man überprüfen, ob alle Verbindungen richtig gesetzt wurden. Dazu wähle ich eine hohe Epsilonumgebung. Ich verwende dazu die „getStringOf“-Methode der Klasse Graph, damit ich mir nur die Postleitzahlen und die Verbindungen anzeigen lassen kann.

```
1 private static void test4() throws IOException
2 {
3     DataManager datenManager = new DataManager();
4     ArrayList<LadeStation> ladeStations = datenManager.leseDatenZeilenweise(new BufferedReader(new
        FileReader(Strings.FILE_NAME_ZUM_TESTEN)));
5     datenManager.sortieren(ladeStations);
6     datenManager.loescheEintraegeInEpsilonUmgebung(ladeStations, MyUtil.intEingabe());
7     Graph graph = new Graph(ladeStations);
8     MyUtil.normalAusgeben(Graph.getStringOf(graph.erzeugeGarphen(MyUtil.intEingabe())));
9 }
```

Auszug aus dem Quellcode - 11

Die Ausgabe enthält das Erwartete. Es wird dem Nutzer angezeigt, welche Ladestationen als Knoten vorhanden sind und die Verbindung zu anderen Ladestationen.

Bei einer Epsilonumgebung von 100 und einer maximalen Entfernung von 200 kommt folgende Ausgabe.



```
"C:\Program Files\Java\jdk-18.0.2\bin\java.exe" -Dvisualvm.id=721861182876000 "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.2\lib\idea_rt.jar=59514:C:\Program
Files\JetBrains\IntelliJ IDEA 2022.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\philipp
.hennken\workspace\EntwicklungsArbeit\out\production\PROG_EA_Hennken_Philipp view.Main
Es sind 100 Ladestationen mit validen Argumenten in der Liste.
Das Einlesen und Überprüfen der Daten dauerte: 11 Millisekunden
Die Liste wurde nach Postleitzahlen aufsteigend sortiert. Bei gleicher Postleitzahl ist die Ladestation mit der höheren Leistung zuerst in der Liste.
Das Sortieren der Ladestationen dauerte: 0 Millisekunden
100
Das Löschen redundanter Daten dauerte: 0 Millisekunden
200
Das Erstellen des Graphen dauerte: 0 Millisekunden
Hier ist die Liste aller Verbindungen zwischen Ladestationen:
79206 --> [71065, 88045]
88045 --> [71065, 79206]
71065 --> [79206, 88045]
|
Process finished with exit code 0
```

Abbildung 12: Erstellung des Graphen mit einer Epsilonumgebung 100 und einer maximalen Entfernung 200

Bei einer Epsilonumgebung von 30 und einer maximalen Entfernung von 200 kommt folgende Ausgabe.


```
"C:\Program Files\Java\jdk-18.0.2\bin\java.exe" -Dvisualvm.id=722576249576200 "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.2\lib\idea_rt.jar=65125:C:\Program Files\JetBrains\IntelliJ IDEA 2022.2\bin" -Dfile.encoding=UTF-8 -classpath C:\Users\philipp\henken\workspace\EntwicklungsArbeit\out\production\PROG_EA_Hennken_Philipp view.Main
Es sind 100 Ladestationen mit validen Argumenten in der Liste.
Das Einlesen und überprüfen der Daten dauerte: 11 Millisekunden
Die Liste wurde nach Postleitzahlen aufsteigend sortiert. Bei gleicher Postleitzahl ist die Ladestation mit der höheren Leistung zuerst in der Liste.
Das Sortieren der Ladestationen dauerte: 0 Millisekunden
Das Löschen redundanter Daten dauerte: 0 Millisekunden
Das Erstellen des Graphen dauerte: 0 Millisekunden
Hier ist die Liste aller Verbindungen zwischen Ladestationen:
72535 --> [71065, 79206, 88045, 88400]
79206 --> [71065, 72535, 88045, 88400]
88045 --> [71065, 72535, 79206, 88400]
71065 --> [72535, 79206, 88045, 88400]
88400 --> [71065, 72535, 79206, 88045]
Process finished with exit code 0
```

Abbildung 13: Erstellung des Graphen mit einer Epsilonumgebung 30 und einer maximalen Entfernung 200

7.5. Test des Aufgabenteil V

Zum Testen der Funktionalität der Wegsuche zwischen zwei Postleitzahlen verwende ich das komplette Programm, so wie es vom Benutzer genutzt werden kann. Dieses beinhaltet alle bereits getesteten Aufgabenteile I bis IV und die Wegsuche.

```
1 private static void programmKomplett() throws IOException
2 {
3     DataManager datenManager = new DataManager();
4     ArrayList<Ladestation> ladestations = datenManager leseDatenZeilenweise(new BufferedReader(new
    FileReader(Strings.FILE_NAME)));
5     datenManager.sortieren (ladestations);
6     epsilonumgebung (ladestations, datenManager);
7     MyUtil.gebeAnbieterUndPlzUndAnschlussleitungAus(ladestations);
8     graphErstellenUndAuswerten(new Graph(ladestations));
9 }
```

Auszug aus dem Quellcode - 12

Die Methode wird auf mehrere Möglichkeiten getestet. Es besteht die Möglichkeit eines vorhandenen Weges, dieser kann beliebig viele Zwischenstationen haben. Dazu wähle ich die Epsilonumgebung 200 und die maximale Entfernung zwischen zwei Ladesäulen auf 250, damit nicht jede Ladestation von jeder anderen erreichbar ist. Ich sehe in den angezeigten Graphen und suche nach einer Ladestation, die nur über mehrere Ecken erreichbar ist. Von der Ladestation in der Postleitzahl 1067 ist die Ladestation in der Postleitzahl 25924 beispielsweise über die Postleitzahlen: 1067 -> 6502 -> 40470 -> 23188 -> 25942.

```
Bitte die Startpostleitzahl eingeben
1067
Bitte die Zielpostleitzahl eingeben
25924
Ein Weg von Edewecht nach Roden ist vorhanden.
Die Suche nach einem Weg dauerte: 2 Millisekunden
```

Abbildung 14: Ein Weg wurde gefunden

Um zu überprüfen, ob auch fehlende Wege als fehlend angezeigt werden, wähle ich die Parameter 200 für die Epsilonumgebung und 220 als maximale Entfernung zwischen zwei Ladesäulen.

Da man mit einer Reichweite von 220 keine andere Ladestation von der Ladestation 70188 erreichen kann, kann dorthin auch kein Weg gefunden werden. Das Programm gibt dies auch an den Nutzer zurück.

```
54291 --> []
6346 --> [40470]
6502 --> [1067]
18230 --> [16230, 25924]
25924 --> [18230, 23188]
23188 --> [25924]
16230 --> [1067, 18230]
70188 --> []
40470 --> [6346]
1067 --> [6502, 16230]
Bitte die Startpostleitzahl eingeben
1067
Bitte die Zielpostleitzahl eingeben
70188
Ein Weg von Thale nach Stuttgart ist nicht vorhanden.
```

Abbildung 15: Es wurde kein Weg gefunden

7.6. Test der kompletten Software

Um die komplette Software zu testen, werden fehlerhafte Eingaben gemacht. Dazu gehören negative Zahlen für die Epsilonumgebung und den maximalen Abstand zwischen Ladestationen. Auch Zeichenketten und Kommazahlen werden, anstatt von ganzen Zahlen, als Eingabe getestet. Ebenfalls muss überprüft werden, ob die vom Nutzer eingegebene Start- und Zielpostleitzahl korrekt ist und sich im Graphen befindet.

7.6.1. Test für die Epsilonumgebung

```
Das Einlesen und überprüfen der Daten dauerte: 161 Millisekunden
Die Liste wurde nach Postleitzahlen aufsteigend sortiert. Bei gleicher Postleitzahl ist die Ladestation mit der höheren Leistung
zuerst in der Liste.
Das Sortieren der Ladestationen dauerte: 7 Millisekunden
Eingabe für Epsilonumgebung
000
Die Epsilonumgebung muss eine ganze Zahl sein, die größer als 0 ist!
Eingabe für Epsilonumgebung
1.0
Eingabe für Epsilonumgebung
130
Die Epsilonumgebung muss eine ganze Zahl sein, die größer als 0 ist!
130
Die Epsilonumgebung muss eine ganze Zahl sein, die größer als 0 ist!
Eingabe für Epsilonumgebung
130
Das Löschen redundanter Daten dauerte: 55 Millisekunden
1: SachsenEnergie AG; Postleitzahl: 1067; Anschlussleistung: 175.0 Kilowatt
2: EVH GmbH; Postleitzahl: 6108; Anschlussleistung: 44.0 Kilowatt
3: Lidl Dienstleistung GmbH & Co. KG; Postleitzahl: 6346; Anschlussleistung: 44.0 Kilowatt
4: envia Mitteldeutsche Energie AG; Postleitzahl: 8626; Anschlussleistung: 44.0 Kilowatt
```

Abbildung 16: Fehlerhafte Eingabe der Epsilonumgebung

Die grüne Schrift ist die Eingabe vom Nutzer.

Es ist zu erkennen, dass der Nutzer bei einer fehlerhaften Eingabe dazu aufgefordert wird, eine korrekte Eingabe zu tätigen. Die Kriterien einer korrekten Eingabe werden dem Nutzer angegeben.

7.6.2. Test für die Entfernung zwischen zwei Ladestationen

```
Welche Reichweite hat das Auto?
31
Die Reichweite des Autos muss eine Zahl in Kilometer sein! Bitte nicht die Einheit angeben!
Welche Reichweite hat das Auto?
-12
Die Reichweite des Autos muss eine Zahl in Kilometer sein! Bitte nicht die Einheit angeben!
Welche Reichweite hat das Auto?
3,5
Die Reichweite des Autos muss eine Zahl in Kilometer sein! Bitte nicht die Einheit angeben!
Welche Reichweite hat das Auto?
150
Das Erstellen des Graphen dauerte: 0 Millisekunden
Hier ist die Liste aller Verbindungen zwischen Ladestationen:
40210 --> [33181, 33181]
21762 --> [19217, 25999, 26757, 29313]
25999 --> [21762]
```

Abbildung 17: Fehlerhafte Eingabe der Reichweite

Bei der Reichweite des Autos, die gleich der maximalen Distanz zwischen zwei Ladesäulen ist, wird eine Fehlerhafte ebenfalls erkenntlich gemacht und der Nutzer kann die Eingabe korrigieren. Erst bei einer korrekten Eingabe wird probiert ein Graph zu erzeugen.

7.6.3. Test für die Eingabe der Postleitzahlen

Für die Eingabe der Postleitzahlen werden sowohl falsche Eingabewerte wie Zeichenketten und Kommazahlen getestet, sowie Postleitzahlen, die insgesamt nicht existieren und valide Postleitzahlen, die allerdings keiner Ladestation im Graphen zugeordnet werden kann.

Zuerst wird auf eine Zeichenkette als Postleitzahl geprüft. Dafür wird die Zeichenkette „bjl“ eingegeben.

```
Bitte die Startpostleitzahl eingeben
bjl
Es lag eine fehlerhafte Eingabe vor. Die eingegebenen Postleitzahlen müssen in der folgenden Liste auftauchen.
Das Erstellen des Graphen dauerte: 0 Millisekunden
Gültige Postleitzahlen sind: Hier ist die Liste aller Verbindungen zwischen Ladestationen:
40210 --> [33181, 33181]
21762 --> [19217, 25999, 26757, 29313]
25999 --> [21762]
54291 --> [83471, 90431]
```

Abbildung 18: Eingabe nicht-valider Postleitzahlen

Es wird ausgegeben, dass eine fehlerhafte Eingabe vorlag. Dem Nutzer werden alle möglichen Eingaben angezeigt und er wird aufgefordert, eine neue Eingabe zu tätigen. Die gleiche Ausgabe erscheint bei der Eingabe einer Kommazahl oder einer negativen Zahl.

Bei der Eingabe einer Ganzzahl wird überprüft, ob die Zahl einer Ladestation im Graphen zugeordnet werden kann. Wenn nicht, erscheint folgende Ausgabe und der Nutzer wird aufgefordert, eine andere, sich in der Liste befindliche, Postleitzahl einzugeben.

```
Bitte die Startpostleitzahl eingeben
800
Bitte die Zielpostleitzahl eingeben
540
Es lag eine fehlerhafte Eingabe vor. Die eingegebenen Postleitzahlen müssen in der folgenden Liste auftauchen.
Das Erstellen des Graphen dauerte: 0 Millisekunden
Gültige Postleitzahlen sind: Hier ist die Liste aller Verbindungen zwischen Ladestationen:
40210 --> [33181, 33181]
21762 --> [19217, 25999, 26757, 29313]
25999 --> [21762]
54291 --> [83471, 90431]
```

Abbildung 19: Fehlerhafte Eingabe einer Postleitzahl die sich nicht im Graphen befindet

Nur bei einer Eingabe von zwei Postleitzahlen wird das Programm ausgeführt.

```
79189 --> [23795, 72669]
63762 --> [83471]
Bitte die Startpostleitzahl eingeben
63762
Bitte die Zielpostleitzahl eingeben
79189
Ein Weg von Halle (Saale) nach Bad Krotzingen ist nicht vorhanden.

Process finished with exit code 0
```

Abbildung 20: Korrekte Eingabe von Postleitzahlen

8. Anwendungsbeispiel

Im Anwendungsbeispiel möchte der Nutzer wissen, ob er von seinem Heimatort 1067 Dresden nach 6268 Querfurt fahren kann, wenn sein Auto eine Reichweite von 250 Kilometern hat. Er möchte allerdings mindestens 150 Kilometer fahren, bevor er hält, weswegen er eine Epsilonumgebung von 150 Kilometer wählt.

Seine Eingaben sind also 150 für die Epsilonumgebung, 250 für die Reichweite des Autos, 1067 als Startpostleitzahl und 6268 als Zielpostleitzahl.

Nachdem alle Ladestationen, die auf Grund von fehlerhaften Geokoordinaten nicht eingetragen wurden, ausgegeben wurden, sieht die Ausgabe auf der Konsole folgendermaßen aus.

```
Das Einlesen und Überprüfen der Daten dauerte: 378 Millisekunden
Die Liste wurde nach Postleitzahlen aufsteigend sortiert. Bei gleicher Postleitzahl ist die Ladestation mit der höheren Leistung zuerst in der Liste.
Das Sortieren der Ladestationen dauerte: 19 Millisekunden
Eingabe für Epsilonumgebung
>E>
Das Löschen redundanter Daten dauerte: 52 Millisekunden
1: SachsenEnergie AG; Postleitzahl: 1067; Anschlussleistung: 175.0 Kilowatt
2: envia Mitteldeutsche Energie AG; Postleitzahl: 6268; Anschlussleistung: 44.0 Kilowatt
3: Lidl Dienstleistung GmbH & Co. KG; Postleitzahl: 6346; Anschlussleistung: 44.0 Kilowatt
4: Gemeinde Litzendorf; Postleitzahl: 9505; Anschlussleistung: 44.0 Kilowatt
5: innogy eMobility Solutions GmbH; Postleitzahl: 10115; Anschlussleistung: 44.0 Kilowatt
6: Stadtwerke Demmin GmbH; Postleitzahl: 17109; Anschlussleistung: 44.0 Kilowatt
7: Charge-ON; Postleitzahl: 19246; Anschlussleistung: 93.0 Kilowatt
8: EWE Go GmbH; Postleitzahl: 23188; Anschlussleistung: 50.0 Kilowatt
9: ENBW mobility+ AG und Co.KG ; Postleitzahl: 23795; Anschlussleistung: 93.0 Kilowatt
10: Charge-ON; Postleitzahl: 24395; Anschlussleistung: 44.0 Kilowatt
11: energcity AG; Postleitzahl: 30521; Anschlussleistung: 22.0 Kilowatt
12: stadtraum Gesellschaft für Raumplanung, Städtebau & Verkehrstechnik mbH; Postleitzahl: 31515; Anschlussleistung: 22.0 Kilowatt
13: Stadtwerke Kevelaer; Postleitzahl: 47623; Anschlussleistung: 22.0 Kilowatt
14: SWT Parken GmbH; Postleitzahl: 54291; Anschlussleistung: 44.0 Kilowatt
15: innogy SE; Postleitzahl: 54326; Anschlussleistung: 44.0 Kilowatt
16: Charge-ON; Postleitzahl: 63762; Anschlussleistung: 22.0 Kilowatt
17: Energiedienst Holding AG; Postleitzahl: 79258; Anschlussleistung: 44.0 Kilowatt
Welche Reichweite hat das Auto?
>R>
Das Erstellen des Graphen dauerte: 0 Millisekunden
Hier ist die Liste aller Verbindungen zwischen Ladestationen:
9505 --> [1067, 6268, 6346, 23795, 54291, 63762]
30521 --> [6268, 6346, 10115, 19246, 23188, 31515]
54291 --> [9505, 23795, 63762]
31515 --> [6268, 6346, 23188, 30521, 47623, 54326]
63762 --> [9505, 54291]
6268 --> [1067, 6346, 9505, 10115, 19246, 30521, 31515]
47623 --> [6346, 31515, 54326]
6346 --> [6268, 9505, 23795, 30521, 31515, 47623, 54326]
54326 --> [6346, 23795, 31515, 47623, 79258]
10115 --> [1067, 6268, 17109, 19246, 30521]
24395 --> [17109, 19246, 23188]
23188 --> [19246, 24395, 30521, 31515]
79258 --> [23795, 54326]
23795 --> [6346, 9505, 54291, 54326, 79258]
19246 --> [6268, 10115, 17109, 23188, 24395, 30521]
17109 --> [10115, 19246, 24395]
1067 --> [6268, 9505, 10115]
Bitte die Startpostleitzahl eingeben
>S>
Bitte die Zielpostleitzahl eingeben
>Z>
Ein Weg von Dresden nach Querfurt ist vorhanden.
Die Suche nach einem Weg dauerte: 9 Millisekunden
Process finished with exit code 0
```

Abbildung 21: Anwendungsbeispiel

9. Verweise

Hier findet sich eine Übersicht über alle Verweise.

9.1. Auszüge aus dem Quellcode

[Auszug aus dem Quellcode – 1: Paket: „model“, Klasse: „Ladestation“, Konstruktor](#)

[Auszug aus dem Quellcode – 2: Paket: „control“, Klasse: „DatenManager“, Methode: „leseDatenZeilenweise“](#)

[Auszug aus dem Quellcode – 3: Paket: „control“, Klasse: „DatenManager“, Methode: „sortieren“](#)

[Auszug aus dem Quellcode – 4: Paket: „control“, Klasse: „DatenManager“, Methode: „loescheEintraegeInEpsilonUmgebung“](#)

Auszug aus dem Quellcode – 5: Paket: „model“, Klasse: „Graph“, Methode:
„erzeugeGraphen“.

Auszug aus dem Quellcode – 6: Paket: „control“, Klasse: „DatenManager“, Methode:
„erhalteLadeStationVonPostleitzahl“.

Auszug aus dem Quellcode – 7: Paket: „model“, Klasse: „WegFinder“, Methode:
„findeWeg“.

Auszug aus dem Quellcode – 8: Paket: „view“, Klasse: „Main“, Methode: „test1“.

Auszug aus dem Quellcode – 9: Paket: „view“, Klasse: „Main“, Methode: „test2“.

Auszug aus dem Quellcode – 10: Paket: „view“, Klasse: „Main“, Methode: „test3“.

Auszug aus dem Quellcode – 11: Paket: „view“, Klasse: „Main“, Methode: „test4“.

Auszug aus dem Quellcode – 12: Paket: „view“, Klasse: „Main“, Methode:
„programmKomplett“

9.2. Abbildungsverzeichnis:

Abbildung 1: UML- Klassendiagramm einer Ladestation.

Abbildung 2: UML-Diagramm der Software.

Abbildung 3: Auszug aus der CSV-Datei.

Abbildung 4: Ausgabe einer nicht eingetragenen Ladestation.

Abbildung 5: Ladestation an der Stelle 18.389.

Abbildung 6: Konsolenausgabe Teilaufgabe I.

Abbildung 7: Ausgabe der unsortierten Liste.

Abbildung 8: Ausgabe der sortierten Liste.

Abbildung 9: Das Programm fordert eine Eingabe.

Abbildung 10: Löschen mit einer Epsilonumgebung von 30.

Abbildung 11: Löschen mit einer Epsilonumgebung von 100.

Abbildung 12: Erstellung des Graphen mit einer Epsilonumgebung 100 und einer

[maximalen Entfernung 200.](#)

[Abbildung 13: Erstellung des Graphen mit einer Epsilonumgebung 30 und einer maximalen Entfernung 200.](#)

[Abbildung 14: Ein Weg wurde gefunden.](#)

[Abbildung 15: Es wurde kein Weg gefunden.](#)

[Abbildung 16: Fehlerhafte Eingabe der Epsilonumgebung](#)

[Abbildung 17: Fehlerhafte Eingabe der Reichweite](#)

[Abbildung 18: Eingabe nicht-valider Postleitzahlen](#)

[Abbildung 19: Fehlerhafte Eingabe einer Postleitzahl die sich nicht im Graphen befindet](#)

[Abbildung 20: Korrekte Eingabe von Postleitzahlen](#)

[Abbildung 21: Anwendungsbeispiel](#)

9.3. Onlineverzeichnis

[Link zu esriDeutschland: https://www.esri.de/de-de/home](https://www.esri.de/de-de/home). Zuletzt aufgerufen am 23.12.2022.

[Link zur Aufgabenstellung für die Entwicklungsarbeit: https://aulis.hs-bremen.de/goto.php?target=file_1772446_download&client_id=hsbremen](https://aulis.hs-bremen.de/goto.php?target=file_1772446_download&client_id=hsbremen). Zuletzt aufgerufen am 25.01.2023.

[Link zum Wikipedia Artikel über Geographische Koordinaten:](#)

https://de.wikipedia.org/wiki/Geographische_Koordinaten. Zuletzt aufgerufen am 29.12.2022.

9.4. Tabellenverzeichnis

[Tabelle 1: Tabelle mit gemessenen Zeiten](#)

10. Anhang

Angehängt sind der gesamte Quellcode und die Javadoc-Dokumentation.

10.1. Quellprogramm

Hier ist der gesamte Quellcode, nach Paketen und Klassen sortiert, abgebildet.

10.1.1. Control Package

10.1.1.1. DatenManager Klasse

```
1. package control;
2.
3. import control.ZeitMessung;
4. import model.Graph;
5. import model.LadeStation;
6. import res.Strings;
7. import util.MyUtil;
8. import java.io.BufferedReader;
9. import java.io.IOException;
10. import java.util.ArrayList;
11.
12. /**      In dieser Klasse können Daten eingelesen, sortiert und
      Graphen erstellt sowie ausgewertet werden.
13. * @author      Philipp Hennken
14. * @version      18.0.2
15. */
16. public class DatenManager
17. {
18.     private int anzahlEitraege = 0;
19.
20.     public int getAnzahlEitraege()
21.     {
22.         return anzahlEitraege;
23.     }
24.
25.     /**      Diese Methode fügt Objekte der Klasse "LadeStation" zu
      einer ArrayList hinzu und löscht anschließend alle Ladestationen heraus,
      deren Geo-Koordinaten nicht möglich sind.
26.     *      Zusätzlich gibt die Methode die Bearbeitungszeit, sowie die
      Anzahl der eingelesenen Dateien aus.
27.     * @pre      Die CSV-Datei, die ausgelesen wird, ist mit Simikola
      getrennt und beinhaltet alle Attribute einer Ladestation.
28.     * @post     Die Ladestationen in der ArrayList haben valide Geo-
      Koordinaten.
29.     * @param    reader BufferedReader der Daten aus einer Datei Zeilenweise
      einliest.
30.     * @return   ArrayList mit Objekten des Typs "LadeStation" und korrekten
      Geo-Koordinaten
31.     */
32.     public ArrayList<LadeStation> leseDatenZeilenweise(BufferedReader
      reader)
33.     {
34.         ZeitMessung.start();
35.         String line = null;
36.         ArrayList<LadeStation> arrayList = new ArrayList<>();
37.         try
38.         {
```

```
39.         while ((line = reader.readLine()) != null) //Bis der Reader
keine weitere Zeile zum Auslesen hat.
40.         {
41.             arrayList.add(new LadeStation(line));
42.             anzahlEitraege++;
43.         }
44.     }
45.     catch (IOException e)
46.     {
47.
48.     }
49.     for (int i = 0; i < arrayList.size(); i++)
50.     {
51.         if (arrayList.get(i).getStrasse() == null) //Wenn dieser
Ausdruck wahr ist, lag ein Fehler in den Geo-Koordinaten vor. Dann wird dem
Parameter Strasse kein Wert zugewiesen und er ist damit 0.
52.         {
53.             MyUtil.normalAusgeben(Strings.LADESTATION_PRE_ANBIETER +
arrayList.get(i).getBetreiber() + Strings.LADESTATION_POST_ANBIETER);
54.             arrayList.remove(i);
55.             i--;
56.             anzahlEitraege--;
57.         }
58.     }
59.     MyUtil.normalAusgeben(Strings.ES_SIND + getAnzahlEitraege() +
Strings.LADESTATIONEN_EINGETRAGEN); //Informationen über die fertige Liste
werden ausgegeben.
60.     MyUtil.normalAusgeben(Strings.EINLESEN_ZEIT +
ZeitMessung.berechneVergangeneZeitInMillisekunden() +
Strings.EINHEIT_ZEITMESSUNG);
61.     return arrayList;
62. }
63.
64. /**      Diese Methode sortiert eine ArrayList mit Objekten des Typs
"LadeStation" zuerst aufsteigend nach der Postleitzahl.
65.      *      Bei zwei gleichen Postleitzahlen wird absteigend nach der
Anschlussleistung einer Ladesäule sortiert.
66.      * @pre      Eine ArrayList wird mit mindestens 2 Objekten vom Typ
"LadeStation" übergeben.
67.      * @post      Die ArrayList ist aufsteigend nach Postleitzahlen sortiert.
Bei gleicher Postleitzahl wurde absteigend nach der Anschlussleistung
sortiert.
68.      * @param      arrayListZuSortieren Die ArrayList, die sortiert werden
soll.
69.      * @return      Die fertig sortierte ArrayList.
70.      */
71.     public ArrayList<LadeStation> sortieren (ArrayList<LadeStation>
arrayListZuSortieren)
72.     {
73.         ZeitMessung.start();
74.         arrayListZuSortieren.sort((LadeStation o1, LadeStation o2) ->
75.         {
76.             if (o1.getPostleitzahl() == o2.getPostleitzahl()) //wenn die
Postleitzahlen gleich sind, wird die Anschlussleistung als Argument
genommen, sonst reicht die Postleitzahl.
77.             {
78.                 return
o2.getAnschlussleistung().compareTo(o1.getAnschlussleistung());
79.             } else
80.             {
```



```
81.         return
            Integer.valueOf(o1.getPostleitzahl()).compareTo(o2.getPostleitzahl());
82.     }
83. });
84.     MyUtil.normalAusgeben(Strings.AUSGABE_LISTE_WURDE_SORTIERT);
85.     MyUtil.normalAusgeben(Strings.SORTIEREN_ZEIT +
        ZeitMessung.berechneVergangeneZeitInMillisekunden() +
        Strings.EINHEIT_ZEITMESSUNG);
86.     return arrayListZuSortieren;
87. }
88.
89. /**      Diese Methode löscht Objekte des Typs "LadeStation" in der
        Epsilonumgebung einer Referenzstation, die vom User eingegeben wird. Eine
        Vergleichsstation, die sich innerhalb der Epsilonumgebung der
        Referenzstation befindet, wird gelöscht.
90.     * @pre      Eine Epsilonumgebung wurde übergeben, die eine ganze Zahl
        und größer als 0 ist.
91.     * @pre      Eine sortierte Liste wurde übergeben.
92.     * @post      Die sortierte Liste beinhaltet nur Objekte vom Typ
        "LadeStation", die mindestens eine Epsilonumgebung voneinander entfernt
        sind.
93.     * @param      sortierteListe Eine ArrayList die Objekte des Typs
        "LadeStation" enthält und nach der Postleitzahl aufsteigend sortiert ist.
94.     * @param      epsilonUmgebung Die Entfernung zwischen zwei Ladesäulen,
        die mindestens bestehen muss, damit die Vergleichsstation nicht gelöscht
        wird.
95.     * @return      Eine ArrayList, die sortiert ist und nur Objekte vom Typ
        "LadeStation" enthält, die mindestens eine Epsilonumgebung voneinander
        entfernt sind.
96.     */
97.     public ArrayList<LadeStation>
        loescheEintraegeInEpsilonUmgebung(ArrayList<LadeStation> sortierteListe,
        int epsilonUmgebung)
98.     {
99.         ZeitMessung.start();
100.        for (int i = 0; i < sortierteListe.size(); i++)//Es wird
            durch alle Einträge der Liste durchiteriert. Dadurch wird immer eine neue
            Ladestation zur Referenzstation.
101.        {
102.            for (int j = i+1; j < sortierteListe.size(); j++)//Es
                wird durch alle Einträge hinter der Referenzstationen iteriert.
103.            {
104.                if(MyUtil.haversine(sortierteListe.get(i).getBreitengrad(),
                    sortierteListe.get(i).getLaengengrad(),
                    sortierteListe.get(j).getBreitengrad(),
                    sortierteListe.get(j).getLaengengrad()) < epsilonUmgebung) //Wenn die
                    Entfernung der beiden Ladesäulen kleiner ist als die Epsilonumgebung, wird
                    die Vergleichsstation herausgelöscht.
105.                {
106.                    sortierteListe.remove(j);
107.                    j--; //"j" muss zurückgesetzt werden, da durch
                    das Rauslöschen, alle anderen Ladestationen nachrutschen. Sonst würde bei
                    jeder Station die herausgelöscht wird, die nächste Ladestation nicht
                    überprüft werden.
108.                }
109.            }
110.        }
111.        MyUtil.normalAusgeben(Strings.LOESCHEN_ZEIT +
            ZeitMessung.berechneVergangeneZeitInMillisekunden() +
            Strings.EINHEIT_ZEITMESSUNG);
```

```
112.         return sortierteListe;
113.     }
114.
115.     /**
116.      * Falls vorhanden, gibt diese Methode ein Objekt vom
117.      * Typ "LadeStation" mit der Postleitzahl zurück, die ihr übergeben wird.
118.      * @pre Ein Graph beinhaltet als Key ein Objekt des Typs
119.      * "LadeStation" und als Value eine ArrayList mit Objekten des Typs
120.      * "LadeStation".
121.      * @post Es wird entweder ein Objekt des Typs "LadeStation"
122.      * oder null übergeben.
123.      * @param postleitzahl Die Postleitzahl, die das Objekt vom Typ
124.      * "LadeStation" haben soll.
125.      * @param graph Der Graph, in dem das Objekt vom Typ
126.      * "LadeStation" gesucht werden soll.
127.      * @return Falls vorhanden, das Objekt vom Typ "LadeStation" mit
128.      * der eingegebenen Postleitzahl, sonst null.
129.      */
130.     public static LadeStation erhalteLadeStationVonPostleitzahl (int
131.     postleitzahl, Graph graph)
132.     {
133.         for (LadeStation ladestation : graph.getGraphKeys()) //Alle
134.             Ladestationen des Graphen werden durchsucht.
135.         {
136.             if (ladestation.getPostleitzahl() == postleitzahl) //Wenn
137.                 die Postleitzahl der Ladestation gleich der gesuchten Postleitzahl ist,
138.                 wurde eine passende Ladestation gefunden.
139.             {
140.                 return ladestation;
141.             }
142.         }
143.         return null;
144.     }
145.
146.
147.
148.
149.
150. }
```

10.1.1.2. ZeitMessung Klasse

```
1. package control;
2.
3. import res.MeineKonstante;
4.
5. /** In dieser Klasse wird die Zeitmessung der einzelnen
6.     * Methoden gesteuert.
7.     * @author Philipp Hennken
8.     * @version 18.0.2
9.     */
10. public class ZeitMessung
11. {
12.     private static long startTime = System.nanoTime();
13.
14.     /** Diese Methode startet die Zeitmessung. Dafür wird die
15.         * aktuelle Zeit in Nanosekunden als Startzeit gesetzt.
16.         * @pre //TODO
17.         * @post Die Zeitmessung wurde gestartet.
```

```
16.    */
17.    public static void start()
18.    {
19.        startTime = System.nanoTime();
20.    }
21.
22.    /**      Diese Methode berechnet die Zeit, die seit dem Start der
23.    *          Zeitmessung vergangen ist.
24.    *          Dafür wird die Differenz der jetzigen Zeit und der Zeit zum
25.    *          Startzeitpunkt genutzt.
26.    *          Die Zahl wird durch 1000 dividiert, damit die Zeit in
27.    *          Millisekunden angegeben werden kann.
28.    * @pre      Die Zeitmessung muss bereits gestartet haben.
29.    * @post      Die vergangene Zeit wurde dem Benutzer ausgegeben.
30.    * @return     Die Zeit die seit dem Start der Zeitmessung vergangen ist
31.    *             in Millisekunden.
32.    */
33.    public static long berechneVergangeneZeitInMillisekunden()
34.    {
35.        return (System.nanoTime() - startTime)/
36.            MeineKonstante.NANO_TO_MILLI;
37.    }
38. }
```

10.1.1.3. Package-info.java

```
1.  /**
2.   * Dieses Paket enthält Klassen zum Verarbeiten von Daten.
3.   * @author Philipp Hennken
4.   * @version 18.0.2
5.   */
6.  package control;
```

10.1.2. Model Package

10.1.2.1. Graph Klasse

```
1.  package model;
2.
3.  import control.ZeitMessung;
4.  import res.Strings;
5.  import util.MyUtil;
6.  import java.util.ArrayList;
7.  import java.util.HashMap;
8.  import java.util.Map;
9.  import java.util.Set;
10.
11. /**      In dieser Klasse wird ein Graph erstellt und verwaltet.
12.  * @author      Philipp Hennken
13.  * @version      18.0.2
14.  */
15. public class Graph
16. {
17.     private final ArrayList<LadeStation> arrayListMitLadeStationen;
```

```
18.     private HashMap<LadeStation, ArrayList<LadeStation>> knoten = new
        HashMap<>();
19.
20.     /**          Dieser Konstruktor weist die übergebene ArrayList der
        internen zu.
21.     * @pre          Eine ArrayList mit Objekten vom Typ "LadeStation" wurde
        übergeben.
22.     * @post         Die interne ArrayList hat einen Verweis auf die
        übergebene ArrayList mit Objekten vom Typ "LadeStation".
23.     * @param arrayListMitLadeStationen Die übergebene ArrayList mit
        Objekten vom Typ "LadeStation".
24.     */
25.     public Graph(ArrayList<LadeStation> arrayListMitLadeStationen)
26.     {
27.         this.arrayListMitLadeStationen = arrayListMitLadeStationen;
28.     }
29.
30.     /**          Diese Methode erzeugt einen Graphen mit einem Objekt des
        Typs "LadeStation" als Key und einer ArrayList mit Objekten des Typs
        "LadeStation" als Value.
31.     *          Die Epsilonumgebung beschreibt den Mindestabstand zwischen
        zwei Ladesäulen, sodass sie gelöscht werden, wenn sie zu nah aneinander
        stehen.
32.     * @pre          Eine Epsilonumgebung wurde übergeben.
33.     * @post         Ein Graph wurde erstellt und beinhaltet nur Objekte vom Typ
        "LadeStation" die mindestens eine Epsilonumgebung voneinander entfernt
        sind.
34.     * @param reichweiteAuto Der Mindestabstand, der zwischen zwei
        Ladesäulen sein muss, damit eine der beiden nicht gelöscht wird.
35.     * @return        Eine HashMap mit einem Objekt des Typs "LadeStation" als
        Key und einer ArrayList mit Objekten des Typs "LadeStation" als Value.
36.     */
37.     public HashMap<LadeStation, ArrayList<LadeStation>> erzeugeGarphen (int
        reichweiteAuto)
38.     {
39.         ZeitMessung.start();
40.         HashMap<LadeStation, ArrayList<LadeStation>> graph = new
        HashMap<>();
41.         for (int i = 0; i < arrayListMitLadeStationen.size(); i++) //Diese
        Schleife ist zum Setzen der Keys.
42.         {
43.             ArrayList<LadeStation> vorruebergendeArrayList = new
        ArrayList<>();
44.             for (int j = 0; j < arrayListMitLadeStationen.size(); j++)
        //Diese Schleife ist zum Setzen der Values.
45.             {
46.                 if(j==i) //Es muss verhindert werden, dass Key und Value
        die gleiche Ladestation ist.
47.                 {
48.                     continue;
49.                 }
50.
51.                 if (reichweiteAuto >
        MyUtil.haversine(arrayListMitLadeStationen.get(i).getBreitengrad(),
        arrayListMitLadeStationen.get(i).getLaengengrad(),
        arrayListMitLadeStationen.get(j).getBreitengrad(),
        arrayListMitLadeStationen.get(j).getLaengengrad()))
52.                 {
53.                     vorruebergendeArrayList.add(arrayListMitLadeStationen
        .get(j));
```

```
54.         }
55.     }
56.     graph.put(arrayListMitLadeStationen.get(i),
    vorruebergendeArrayList);
57.
58.     }
59.     this.setKnoten(graph);
60.     MyUtil.normalAusgeben(Strings.GRAPH_ERSTELLEN +
    ZeitMessung.berechneVergangeneZeitInMillisekunden() +
    Strings.EINHEIT_ZEITMESSUNG);
61.     return graph;
62. }
63.
64. /**      Diese Methode gibt einen String für eine schöne Ausgabe
    einer HashMap zurück.
65.     * @pre      Eine HashMap mit einem Objekt des Typs "LadeStation" als
    Key und einer ArrayList mit Objekten des Typs "LadeStation" als Value wurde
    übergeben.
66.     * @post      Ein String steht zur Ausgabe bereit.
67.     * @param      ladeStationArrayListHashMap Die HashMap die ausgegeben
    werden soll.
68.     * @return      En String der eine HashMap mit einem Objekt des Typs
    "LadeStation" als Key und einer ArrayList mit Objekten des Typs
    "LadeStation" als Value schön anzeigt.
69.     */
70.     public static String getStringOf(HashMap<LadeStation,
    ArrayList<LadeStation>> ladeStationArrayListHashMap) {
71.         if (ladeStationArrayListHashMap == null)    {
72.             return "";
73.         }
74.         StringBuilder b = new StringBuilder();
75.         b.append(Strings.LISTE_ALLER_VERBINDUNGEN);
76.
77.         Set<Map.Entry<LadeStation, ArrayList<LadeStation>>> set =
    ladeStationArrayListHashMap.entrySet();
78.         for (Map.Entry entry : set) {
79.             Object key = entry.getKey();
80.             Object value = entry.getValue();
81.
82.             b.append(Strings.ABSATZ);
83.             b.append(key);
84.             b.append(Strings.PFEIL);
85.             b.append(value);
86.         }
87.         return b.toString();
88.     }
89.
90.     public HashMap<LadeStation, ArrayList<LadeStation>> getGraph()
91.     {
92.         return knoten;
93.     }
94.
95.     /**      Diese Methode gibt alle Keys eines Graphen wieder.
96.     * @pre      Eine HashMap mit Keys wurde dem Set mit Objekten des Typs
    "LadeStation" namens "knoten" zugewiesen.
97.     * @post      Alle Keys des Graphen wurden zurückgegeben.
98.     * @return      Alle Keys des Graphen.
99.     */
100.    public Set<LadeStation> getGraphKeys()
101.    {
```

```
102.         return knoten.keySet();
103.     }
104.
105.     public void setKnoten(HashMap<LadeStation,
        ArrayList<LadeStation>> knoten)
106.     {
107.         this.knoten = knoten;
108.     }
109. }
```

10.1.2.2. LadeStation Klasse

```
1. package model;
2.
3. import res.Strings;
4. import res.MeineKonstante;
5.
6. /**          In dieser Klasse werden alle relevanten Daten einer
    Ladestation abgespeichert und verwendbar gemacht.
7.  * @author    Philipp Hennken
8.  * @version    18.0.2
9.  */
10. public class LadeStation
11. {
12.     private final String betreiber;
13.     private String strasse;
14.     private String hausnummer;
15.     private int postleitzahl;
16.     private String ort;
17.     private String bundesland;
18.     private float breitengrad;
19.     private float laengengrad;
20.     private float anschlussleistung;
21.
22.
23.     /**          In diesem Konstruktor werden Objekte vom Typ "LadeStation"
    erstellt und die relevanten Attribute zugewiesen.
24.     * @pre      String is not empty.
25.     * @post     Relevante Attribute wurden zugewiesen.
26.     * @param    input Eine eingelesene Zeile der auszuwertenden CSV-Datei,
    die durch Simikola getrennt sind.
27.     */
28.     public LadeStation (String input)
29.     {
30.         String[] splits = input.split(Strings.SIMIKOLON);
31.         if(          //Bedingungen für korrekte Geokoordinaten.
32.             Float.parseFloat(splits[MeineKonstante.BREITENGRAD_INDEX])
33.             > 0 &&
34.             Float.parseFloat(splits[MeineKonstante.BREITENGRAD_INDEX])
35.             < 90 &&
36.             Float.parseFloat(splits[MeineKonstante.LAENGENGRAD_INDEX])
37.             > 0 &&
38.             Float.parseFloat(splits[MeineKonstante.LAENGENGRAD_INDEX])
39.             < 180
```

```
39.         this.betreiber = splits[MeineKonstante.BETREIBER_INDEX];
40.         this.strasse = splits[MeineKonstante.STRASSE_INDEX];
41.         this.hausnummer = splits[MeineKonstante.HAUSNUMMER_INDEX];
42.         this.postleitzahl =
    Integer.parseInt(splits[MeineKonstante.POSTLEITZAHL_INDEX]);
43.         this.ort = splits[MeineKonstante.ORT_INDEX];
44.         this.bundesland = splits[MeineKonstante.BUNDESLAND_INDEX];
45.         this.breitengrad =
    Float.parseFloat(splits[MeineKonstante.BREITENGRAD_INDEX]);
46.         this.laengengrad =
    Float.parseFloat(splits[MeineKonstante.LAENGENGRAD_INDEX]);
47.         this.anschlussleistung =
    Float.parseFloat(splits[MeineKonstante.ANSCHLUSSLEISTUNG_INDEX]);
48.     }else
49.     { //Bei Fehlerhaften Geokoordinaten werden nur Ort und Betreiber
    richtig zugewiesen. Später kann mit der Überprüfung bspw. der Strasse
    geschaut werden, ob ein Fehler bei dieser LAdestation vorlag. Wenn einer
    Vorlag ist die Aussage "getStrass()==null" wahr.
50.         this.betreiber = splits[MeineKonstante.BETREIBER_INDEX];
51.         this.ort = splits[MeineKonstante.ORT_INDEX];
52.     }
53. }
54.
55.
56.
57.
58.
59. public String getBetreiber()
60. {
61.     return betreiber;
62. }
63.
64. public String getStrasse()
65. {
66.     return strasse;
67. }
68. public int getPostleitzahl()
69. {
70.     return postleitzahl;
71. }
72.
73. public String getOrt()
74. {
75.     return ort;
76. }
77. public float getBreitengrad()
78. {
79.     return breitengrad;
80. }
81. public float getLaengengrad()
82. {
83.     return laengengrad;
84. }
85. public Float getAnschlussleistung()
86. {
87.     return anschlussleistung;
88. }
89.
90.
91.
```

```
92.     public String toString()
93.     {
94.         return String.valueOf(postleitzahl);
95.     }
96.
97.
98.
99. }
```

10.1.2.3. LadestationNichtInGeaphException Exception

```
1. package model;
2.
3. /**          In dieser Klasse wird eine Exception formuliert.
4.  * @author    Philipp Hennken
5.  * @version    18.0.2
6.  */
7. public class LadestationNichtInGraphException extends Exception
8. {
9.     /**      Der Konstruktor ruft den Konstruktor der Superklasse
10.    "Exception" auf.
11.    */
12.    public LadestationNichtInGraphException()
13.    {
14.        super();
15.    }
16. }
```

10.1.2.4. WegFinder Klasse

```
1. package model;
2.
3. import control.DatenManager;
4. import control.ZeitMessung;
5. import res.MeineKonstante;
6. import res.Strings;
7. import util.MyUtil;
8.
9. import java.util.ArrayList;
10.
11. /**          In dieser Klasse wird nach einem möglichen Weg zwischen
12.    zwei Objekten des Typs "LadeStation" gesucht.
13.    * @author    Philipp Hennken
14.    * @version    18.0.2
15.    */
16. public class WegFinder
17. {
18.     private Graph graph = null;
19.     private int startPLZ = 0;
20.     private int endPLZ = 0;
21.
22.     /**
```



```
23.      *           Diese Methode findet einen Weg von einer Ladestation in dem
      Ort, mit der ersten eingegebenen Postleitzahl zu einer Ladestation in dem
      Ort mit der anderen eingegebenen Postleitzahl.
24.      *           Die Suche des Weges basiert auf einer Tiefensuche.
25.      * @pre       Ein Graph beinhaltet Objekte des Typs "LadeStation" als
      Keys und Values.
26.      * @post      Das Programm wurde beendet, nachdem der Nutzer über die
      Existenz beziehungsweise Nichtexistenz eines Weges informiert wurde.
27.      * @param     graph In diesem Graphen wird nach den Haltestellen für den
      Weg zwischen den Orten gesucht.
28.      * @param     startPLZ Die Postleitzahl, von dem Ort, wo der Weg
      beginnen soll.
29.      * @param     endPLZ Die Postleitzahl, von dem Ort, wo der Weg enden
      soll.
30.      */
31.      public void findeWeg(Graph graph, int startPLZ, int endPLZ,
      ArrayList<LadeStation> besuchteLadeStationen) throws
      LadeStationNichtInGraphException
32.      {
33.          ZeitMessung.start();
34.          this.startPLZ = startPLZ;
35.          this.endPLZ = endPLZ;
36.          this.graph = graph;
37.          testeObStartUndZielPostleitzahlImGraphenSind();
38.          LadeStation startLadeStation =
      DataManager.erhalteLadeStationVonPostleitzahl(startPLZ, graph);
39.          for (LadeStation
      ladeStation:graph.getGraph().get(startLadeStation))
40.          {
41.              if (! besuchteLadeStationen.contains(ladeStation))
42.              {
43.                  if (ladeStation.getPostleitzahl() == endPLZ)
44.                  {
45.                      wegIstVorhanden();
46.                  }else
47.                  {
48.                      besuchteLadeStationen.add(startLadeStation);
49.                      findeWeg(graph, ladeStation.getPostleitzahl(), endPLZ,
      besuchteLadeStationen);
50.                  }
51.              }
52.          }
53.          keinWegVorhanden();
54.
55.      }
56.
57.
58.      /**           Diese Methode gibt aus, dass ein Weg vorhanden ist.
59.      *           Außerdem gibt die Methode den Startort sowie den Zielort
      aus.
60.      * @pre       Ein Weg vom Startort zum Zielort ist vorhanden.
61.      * @post      Dem Nutzer wurde angezeigt, dass ein Weg vorhanden ist. Das
      Programm wird beendet.
62.      */
63.      private void wegIstVorhanden()
64.      {
65.          MyUtil.normalAusgeben(Strings.WEG_VORHANDEN_PRE +
      DataManager.erhalteLadeStationVonPostleitzahl(startPLZ, graph).getOrt() +
      Strings.WEG_VORHANDEN_NACH +
```

```
DatenManager.erhalteLadeStationVonPostleitzahl(endPLZ, graph).getOrt() +
Strings.WEG_VORHANDEN_POST);
66.    MyUtil.normalAusgeben(Strings.WEG_FINDER +
    ZeitMessung.berechneVergangeneZeitInMillisekunden() +
    Strings.EINHEIT_ZEITMESSUNG);
67.    System.exit(MeineKonstante.ZERO);
68. }
69.
70. /**      Diese Methode überprüft, ob ein Objekt vom Typ
    "LadeStation" die Postleitzahl vom Startort und ein Objekt vom Typ
    "LadeStation" die Postleitzahl vom Zielort hat.
71.     * @pre      Startpostleitzahl und Zielpostleitzahl wurden angegeben.
72.     * @post      Wenn keine Exception geworfen wird, passiert nichts.
73.     *            Wenn eine Exception geworfen wird, kann diese gefangen
    werden.
74.     * @throws    LadestationNichtInGraphException wird geworfen, wenn eine
    oder beide Postleitzahlen nicht zugeordnet werden können.
75.     */
76.    private void testeObStartUndZielPostleitzahlImGraphenSind() throws
    LadestationNichtInGraphException
77.    {
78.        if (!
    (graph.getGraphKeys().contains(DatenManager.erhalteLadeStationVonPostleitza
    hl(startPLZ, graph)) &&
    graph.getGraphKeys().contains(DatenManager.erhalteLadeStationVonPostleitzah
    l(endPLZ, graph))))
79.        {
80.            throw new LadestationNichtInGraphException();
81.        }
82.    }
83.
84. /**      Diese Methode gibt an, wenn kein Weg im Graphen zwischen
    Startort und Zielort vorhanden ist.
85.     * @pre      Es ist kein Weg vom Startort zum Zielort vorhanden.
86.     * @post      Der Nutzer wird informiert, dass es kein Weg gibt und das
    Programm wird beendet.
87.     */
88.    private void keinWegVorhanden()
89.    {
90.        MyUtil.fehlerAusgeben(Strings.WEG_VORHANDEN_PRE +
    DatenManager.erhalteLadeStationVonPostleitzahl(startPLZ, graph).getOrt() +
    Strings.WEG_VORHANDEN_NACH +
    DatenManager.erhalteLadeStationVonPostleitzahl(endPLZ, graph).getOrt() +
    Strings.WEG_NICHT_VORHANDEN_POST);
91.        System.exit(MeineKonstante.ZERO);
92.    }
93.
94.
95. }
```

10.1.2.5. Package-info.java

```
1. /**
2.  * Dieses Paket modelliert Datenstrukturen.
3.  * @author Philipp Hennken
4.  * @version 18.0.2
5.  */
```

```
6. package model;
```

10.1.3. Res Package

10.1.3.1. LadeStationen.csv CSV-Datei

10.1.3.2. LadeStationZumTesten.csv CSV-Datei

10.1.3.3. MeineKonstante Interface

```
1. package res;
2.
3. /**          In diesem Interface befinden sich Konstanten, damit der
   Code übersichtlich bleibt.
4.  * @author    Philipp Hennken
5.  * @version    18.0.2
6.  */
7. public interface MeineKonstante
8. {
9.     int NANO_TO_MILLI = 1000000;
10.    int BETREIBER_INDEX = 0;
11.    int STRASSE_INDEX = 1;
12.    int HAUSNUMMER_INDEX = 2;
13.    int POSTLEITZAHL_INDEX = 3;
14.    int ORT_INDEX = 4;
15.    int BUNDESLAND_INDEX = 5;
16.    int BREITENGRAD_INDEX = 6;
17.    int LAENGENGRAD_INDEX = 7;
18.    int ANSCHLUSSLEISTUNG_INDEX = 8;
19.    int ZERO = 0;
20. }
```

10.1.3.4. Strings Interface

```
1. package res;
2.
3.
4. /**          In diesem Interface befinden sich Literale, damit der Code
   übersichtlich bleibt.
5.  * @author    Philipp Hennken
6.  * @version    18.0.2
7.  */
8. public interface Strings
9. {
10.    String SIMIKOLON = ";";
11.    String EINHEIT_ZEITMESSUNG = " Millisekunden";
12.    String EINHEIT_ANACHLUSSLEISTUNG = " Kilowatt";
13.    String LEERZEICHEN = " ";
14.    String LADESTATION_PRE_ANBIETER = "Eine Ladestation von ";
15.    String LADESTATION_POST_ANBIETER = "' wurde nicht hinzugefuegt, da die
   Geokoordinaten fehlerhaft sind.";
16.    String FILE_NAME = "./src/res/LadeStationen.csv";
17.    String FILE_NAME_ZUM_TESTEN = "./src/res/LadeStationenZumTesten.csv";
18.    String START_PLZ_FRAGE = "Bitte die Startpostleitzahl eingeben";
```

```
19.    String END_PLZ_FRAGE = "Bitte die Zielpostleitzahl eingeben";
20.    String FALSCHHE_PLZ_EINGABE = "Es lag eine fehlerhafte Eingabe vor. Die
    eingegebenen Postleitzahlen müssen in der folgenden Liste auftauchen.";
21.    String GÜLTIGE_PLZ_SIND = "Gültige Postleitzahlen sind: ";
22.    String REICHWEITE_FALSCH = "Die Reichweite des Autos muss eine Zahl in
    Kilometer sein! Bitte nicht die Einheit angeben!";
23.    String REICHWEITE_FRAGE = "Welche Reichweite hat das Auto?";
24.    String EPSILONUMGEBUNG_FALSCH = "Die Epsilonumgebung muss eine ganze
    Zahl sein, die größer als 0 ist!";
25.    String EPSILONUMGEBUNG_ABFRAGE = "Eingabe für Epsilonumgebung";
26.    String LISTE_ALLER_VERBINDUNGEN = "Hier ist die Liste aller
    Verbindungen zwischen Ladestationen:";
27.    String PFEIL = " --> ";
28.    String ABSATZ = "\n";
29.    String WEG_VORHANDEN_PRE = "Ein Weg von ";
30.    String DOPPELPUNKT = ": ";
31.    String PLZ = "Postleitzahl: ";
32.    String ANSCHLUSSLEISTUNG = "Anschlussleistung: ";
33.    String SORTIEREN_ZEIT = "Das Sortieren der Ladestationen dauerte: ";
34.    String EINLESEN_ZEIT = "Das Einlesen und Überprüfen der Daten dauerte:
    ";
35.    String LOESCHEN_ZEIT = "Das Löschen redundanter Daten dauerte: ";
36.    String GRAPH_ERSTELLEN = "Das Erstellen des Graphen dauerte: ";
37.    String WEG_FINDER = "Die Suche nach einem Weg dauerte: ";
38.    String WEG_VORHANDEN_NACH = " nach ";
39.    String WEG_VORHANDEN_POST = " ist vorhanden.";
40.    String WEG_NICHT_VORHANDEN_POST = " ist nicht vorhanden.";
41.    String ES_SIND = "Es sind ";
42.    String LADESTATIONEN_EINGETRAGEN = " Ladestationen mit validen
    Argumenten in der Liste.";
43.    String AUSGABE_LISTE_WURDE_SORTIERT = "Die Liste wurde nach
    Postleitzahlen aufsteigend sortiert. Bei gleicher Postleitzahl ist die
    Ladestation mit der höheren Leistung zuerst in der Liste.";
44. }
```

10.1.3.5. Package-info.java

```
1. /**
2.  * Dieses Paket enthält programmspezifische Daten.
3.  * @author Philipp Hennken
4.  * @version 18.0.2
5.  */
6. package res;
```

10.1.4. Util Package

10.1.4.1. MyUtil Klasse

```
1. package util;
2.
3. import model.LadeStation;
4. import res.Strings;
5. import java.io.BufferedReader;
6. import java.io.IOException;
```

```
7. import java.io.InputStreamReader;
8. import java.util.ArrayList;
9.
10. /**          In dieser Klasse werden Ein- und Ausgaben auf der Konsole
    gesteuert und verwaltet.
11. * @author      Philipp Hennken
12. * @version     18.0.2
13. */
14. public class MyUtil
15. {
16.     /**          Diese Methode gibt den Anbieter, die Postleitzahl und die
    Anschlussleistung einer ArrayList von Ladestationen aus.
17.     * @pre       Die ArrayList beinhaltet Objekte des Typs "LadeStation" und
    ist nicht leer.
18.     * @post      Die Daten wurden über die Konsole an den Nutzer ausgegeben.
19.     * @param     daten ArrayList mit Ladestationen.
20.     */
21.     public static void
    gebeAnbieterUndPlzUndAnschlussleistungAus(ArrayList<LadeStation> daten)
22.     {
23.         for (int i = 0; i < daten.size(); i++)
24.         {
25.             MyUtil.normalAusgeben((i+1) + Strings.DOPPELPUNKT +
    daten.get(i).getBetreiber() + Strings.SIMIKOLON + Strings.LEERZEICHEN +
    Strings.PLZ + daten.get(i).getPostleitzahl() + Strings.SIMIKOLON +
    Strings.LEERZEICHEN + Strings.ANSCHLUSSLEISTUNG +
    daten.get(i).getAnschlussleistung() + Strings.EINHEIT_ANACHLUSSLEISTUNG);
26.         }
27.     }
28.
29.     static BufferedReader reader = new BufferedReader(new
    InputStreamReader(System.in));
30.
31.     /**          Diese Methode fordert vom Benutzer eine Eingabe einer ganzen
    Zahl und gibt diese wieder.
32.     * @pre       Der eingegebene Wert ist eine ganze Zahl.
33.     * @post      Die eingegebene Zeile wurde ausgelesen und zurückgegeben.
34.     * @return     Der eingegebene Zahlenwert des Benutzers.
35.     * @throws    IOException
36.     */
37.     public static int intEingabe() throws IOException
38.     {
39.
40.         return Integer.parseInt(reader.readLine());
41.     }
42.
43.     /**
44.     * Diese Methode gibt ein Objekt aus.
45.     * @pre       Das eingegebene Objekt kann ausgegeben werden.
46.     * @post      Die Daten wurden über die Konsole an den Nutzer ausgegeben.
47.     * @param     ausgabeObjekt Das übergebene Objekt, das ausgegeben werden
    soll.
48.     */
49.     public static void normalAusgeben(Object ausgabeObjekt)
50.     {
51.         System.out.println(ausgabeObjekt);
52.     }
53.
54.     /**
55.     * Diese Methode gibt ein Objekt als Fehlermeldung aus.
```

```
56.     * @pre      Das eingegebene Objekt kann ausgegeben werden.
57.     * @post      Die Daten wurden über die Konsole als Fehlermeldung an den
    Nutzer ausgegeben.
58.     * @param      ausgabeObjekt Das übergebene Objekt, das als Fehlermeldung
    ausgegeben werden soll.
59.     */
60.     public static void fehlerAusgeben (Object ausgabeObjekt)
61.     {
62.         System.err.println(ausgabeObjekt);
63.     }
64.
65.
66.     /**
67.     * Diese Methode ermittelt den Abstand zwischen zwei Punkten auf einer
    Kugel anhand von Geokoordinaten.
68.     * Die Methode stammt von der Internetseite
    https://de.acervolima.com/haversine-formel-zum-ermitteln-des-abstands-
    zwischen-zwei-punkten-auf-einer-kugel/.
69.     * @author      PRAKHAR7
70.     * @param      BreitengradEins Breitengrad des ersten Ortes.
71.     * @param      LaengengradEins Längengrad des ersten Ortes.
72.     * @param      BreitengradZwei Breitengrad des zweiten Ortes.
73.     * @param      LaengengradZwei Längengrad des ersten Ortes.
74.     * @return      Abstand zwischen den beiden Orten in Kilometern.
75.     */
76.     public static double haversine(double BreitengradEins, double
    LaengengradEins,
77.                                     double BreitengradZwei, double
    LaengengradZwei)
78.     {
79.         // distance between latitudes and longitudes
80.         double dLat = Math.toRadians(BreitengradZwei - BreitengradEins);
81.         double dLon = Math.toRadians(LaengengradZwei - LaengengradEins);
82.
83.         // convert to radians
84.         BreitengradEins = Math.toRadians(BreitengradEins);
85.         BreitengradZwei = Math.toRadians(BreitengradZwei);
86.
87.         // apply formulae
88.         double a = Math.pow(Math.sin(dLat / 2), 2) +
89.             Math.pow(Math.sin(dLon / 2), 2) *
90.             Math.cos(BreitengradEins) *
91.             Math.cos(BreitengradZwei);
92.         double rad = 6371;
93.         double c = 2 * Math.asin(Math.sqrt(a));
94.         return rad * c;
95.     }
96.
97.
98. }
```

10.1.4.2. Package-info.java

```
1. /**
2.  * Dieses Paket enthält Klassen zum Ein- und Ausgeben.
3.  * @author Philipp Hennken
4.  * @version 18.0.2
```

```
5. */
6. package util;
```

10.1.5. View Package

10.1.5.1. Main Klasse

```
1. package view;
2.
3. import control.DatenManager;
4. import model.*;
5. import res.Strings;
6. import util.MyUtil;
7. import java.io.*;
8. import java.util.ArrayList;
9.
10. /**          In dieser Klasse wird das Programm gestartet.
11.  * @author    Philipp Hennken
12.  * @version    18.0.2
13.  */
14. public class Main
15. {
16.     /**      Diese Methode wird zum Starten des Programms genutzt.
17.     * @pre      Das Programm wurde gestartet.
18.     * @post      Das Programm wurde beendet.
19.     * @param      args Die Programmargumente.
20.     * @throws      IOException Eine IOException kann auftreten, wenn falsche
        Daten vom Nutzer eingegeben werden.
21.     * @throws      LadestationNichtInGraphException Sollte eine Postleitzahl
        eingegeben werden, die keiner Ladestation im Graphen zugeordnet ist, wird
        diese Exception geworfen.
22.     */
23.     public static void main (String[] args) throws IOException,
        LadestationNichtInGraphException
24.     {
25.         programmKomplett();
26.         /*test1();*/
27.         /*test2();*/
28.         /*test3();*/
29.         /*test4();*/
30.     }
31.
32.     /**      Diese Methode dient zum Testen der Aufgaben des ersten
        Aufgabenteils.
33.     * @pre      Das Programm wurde gestartet.
34.     * @post      Das Programm wurde beendet.
35.     * @throws      FileNotFoundException Diese Exception wird geworfen, sollte
        keine Datei im angegebenen Pfad liegen.
36.     */
37.     private static void test1() throws FileNotFoundException
38.     {
39.         DatenManager datenManager = new DatenManager();
40.         ArrayList<Ladestation> ladestations =
        datenManager leseDatenZeilenweise(new BufferedReader(new
        FileReader(Strings.FILE_NAME)));
41.         MyUtil.gebeAnbieterUndPlzUndAnschlussleitungAus(ladestations);
42.     }
```

```
43.  /**          Diese Methode dient zum Testen der Aufgaben des zweiten
    Aufgabenteils.
44.      *@pre      Das Programm wurde gestartet.
45.      *@post      Das Programm wurde beendet.
46.      * @throws  FileNotFoundException Diese Exception wird geworfen, sollte
    keine Datei im angegebenen Pfad liegen.
47.      */
48.  private static void test2() throws FileNotFoundException
49.  {
50.      DataManager datenManager = new DataManager();
51.      ArrayList<LadeStation> ladeStations =
    datenManager leseDatenZeilenweise(new BufferedReader(new
    FileReader(Strings.FILE_NAME_ZUM_TESTEN)));
52.      MyUtil.gebeAnbieterUndPlzUndAnschlussleitungAus(ladeStations);
53.      datenManager.sortieren(ladeStations);
54.      MyUtil.gebeAnbieterUndPlzUndAnschlussleitungAus(ladeStations);
55.  }
56.  /**          Diese Methode dient zum Testen der Aufgaben des dritten
    Aufgabenteils.
57.      *@pre      Das Programm wurde gestartet.
58.      *@post      Das Programm wurde beendet.
59.      * @throws  FileNotFoundException Diese Exception wird geworfen, sollte
    keine Datei im angegebenen Pfad liegen.
60.      */
61.  private static void test3() throws IOException
62.  {
63.      DataManager datenManager = new DataManager();
64.      ArrayList<LadeStation> ladeStations =
    datenManager leseDatenZeilenweise(new BufferedReader(new
    FileReader(Strings.FILE_NAME_ZUM_TESTEN)));
65.      datenManager.sortieren(ladeStations);
66.      datenManager.loescheEintraegeInEpsilonUmgebung(ladeStations,
    MyUtil.intEingabe());
67.      MyUtil.gebeAnbieterUndPlzUndAnschlussleitungAus(ladeStations);
68.  }
69.  /**          Diese Methode dient zum Testen der Aufgaben des vierten
    Aufgabenteils.
70.      *@pre      Das Programm wurde gestartet.
71.      *@post      Das Programm wurde beendet.
72.      * @throws  FileNotFoundException Diese Exception wird geworfen, sollte
    keine Datei im angegebenen Pfad liegen.
73.      */
74.  private static void test4() throws IOException
75.  {
76.      DataManager datenManager = new DataManager();
77.      ArrayList<LadeStation> ladeStations =
    datenManager leseDatenZeilenweise(new BufferedReader(new
    FileReader(Strings.FILE_NAME_ZUM_TESTEN)));
78.      datenManager.sortieren(ladeStations);
79.      datenManager.loescheEintraegeInEpsilonUmgebung(ladeStations,
    MyUtil.intEingabe());
80.      Graph graph = new Graph(ladeStations);
81.      MyUtil.normalAusgeben(Graph.getStringOf(graph.erzeugeGarphen(MyUtil
    .intEingabe())));
82.  }
83.
84.
85.  /**          Diese Methode beinhaltet den kompletten Ablauf des
    Programms.
```



```
86.      *           Diese Methode dient zum Testen der Aufgaben des letzten
Aufgabenteils.
87.      *@pre       Das Programm wurde gestartet.
88.      *@post      Das Programm wurde beendet.
89.      * @throws   FileNotFoundException Diese Exception wird geworfen, sollte
keine Datei im angegebenen Pfad liegen.
90.      */
91.      private static void programmKomplett() throws IOException
92.      {
93.          DataManager datenManager = new DataManager();
94.          ArrayList<LadeStation> ladeStations =
datenManager leseDatenZeilenweise(new BufferedReader(new
FileReader(Strings.FILE_NAME)));
95.          datenManager.sortieren (ladeStations);
96.          epsilonumgebung (ladeStations, datenManager);
97.          MyUtil.gebeAnbieterUndPlzUndAnschlussleitungAus(ladeStations);
98.          graphErstellenUndAuswerten(new Graph(ladeStations));
99.      }
100.
101.      /**           Diese Methode erstellt einen Graphen aus allen
Objekten des Typs "LadeStation", die zuvor eingelesen wurden.
102.      *           Als Key wird im Graph ein Objekt des Typs
"LadeStation" sein und als Value eine ArrayList mit Objekten des Typs
"LadeStation", die von der Key-Ladestation erreichbar sind. Ein Objekt des
Typs "LadeStation" ist erreichbar, wenn die Entfernung kleiner oder gleich
der Reichweite des Autos ist.
103.      *
104.      * @pre       Die Reichweite des Autos ist eine ganze Zahl, die
größer ist als 0.
105.      *           Der Graph besteht aus einem Objekt des Typen
"LadeStation" als Key und einer ArrayList mit Objekten des Typs
"LadeStation" als Value.
106.      * @post      Ein Graph wurde erstellt.
107.      *           Es wurde überprüft, ob ein Weg von der Start-
Postleitzahl zur Ziel-Postleitzahl existiert.
108.      *           Das Ergebnis der Überprüfung wurde dem Nutzer
mitgeteilt.
109.      * @param     graph Ein Graph, der aus der ArrayList aller Objekte
des Typs "LadeStation" besteht.
110.      */
111.      private static void graphErstellenUndAuswerten(Graph graph)
112.      {
113.          int reichweiteAuto;
114.          while(true)
115.          {
116.              try
117.              {
118.                  MyUtil.normalAusgeben(Strings.REICHWEITE_FRAGE);
119.                  reichweiteAuto = MyUtil.intEingabe();
120.                  if(reichweiteAuto <=0)
121.                  {
122.                      throw new NumberFormatException();
123.                  }
124.                  MyUtil.normalAusgeben(Graph.getStringOf(graph.erzeuge
Garphen(reichweiteAuto)));
125.                  break;
126.              } catch (NumberFormatException e)
127.              {
128.                  MyUtil.fehlerAusgeben(Strings.REICHWEITE_FALSCH);
129.              } catch (IOException e)
```

```
130.         {
131.             throw new RuntimeException(e);
132.         }
133.     }
134.     WegFinder wegFinder = new WegFinder();
135.     while (true)
136.     {
137.         try
138.         {
139.             MyUtil.normalAusgeben(Strings.START_PLZ_FRAGE);
140.             int startPLZ = MyUtil.intEingabe();
141.             MyUtil.normalAusgeben(Strings.END_PLZ_FRAGE);
142.             int endPLZ = MyUtil.intEingabe();
143.             wegFinder.findeWeg(graph, startPLZ, endPLZ, new
ArrayList<>());
144.             break;
145.         } catch (LadestationNichtInGraphException |
NumberFormatException e)
146.         {
147.             MyUtil.fehlerAusgeben(Strings.FALSCH_EINGABE);
148.             MyUtil.normalAusgeben( Strings.GUELFIGE_PLZ_SIND +
Graph.getStringOf(graph.erzeugeGraphen(reichweiteAuto)));
149.         } catch (IOException e)
150.         {
151.             throw new RuntimeException(e);
152.         }
153.     }
154. }
155.
156. /**      Diese Methode fragt den Benutzer nach einer
Epsilonumgebung.
157. *      Dann wird die loescheEintraegeInEpsilonUmgebung() in
der Klasse DatenManager aufgerufen.
158. * @pre      Es wird eine nicht leere ArrayList übergeben. Die
einzugebene Epsilonumgebung ist eine ganze Zahl, größer als 0.
159. * @post      Die ArrayList ladestations besteht nur noch aus
Objekten des Typs "Ladestation", die mindestens eine Epsilonumgebung
voneinander entfernt sind.
160. * @param      ladestations ArrayList mit Objekten vom Typ
"Ladestation".
161. * @param      datenManager Objekt vom Typ DatenManager.
162. */
163. private static void epsilonumgebung(ArrayList<Ladestation>
ladestations, DatenManager datenManager)
164. {
165.     while(true)
166.     {
167.         try
168.         {
169.             MyUtil.normalAusgeben(Strings.EPSILONUMGEBUNG_ABFRAGE
);
170.             int epsilonumgebung = MyUtil.intEingabe();
171.             if(epsilonumgebung<=0)
172.             {
173.                 throw new NumberFormatException();
174.             }
175.
176.             datenManager.loescheEintraegeInEpsilonUmgebung(ladeSt
ations, epsilonumgebung);
177.             break;
```

```
178.         }catch (NumberFormatException e)
179.         {
180.             MyUtil.fehlerAusgeben(Strings.EPSILONUMGEBUNG_FALSCH)
181.         ;
182.         } catch (IOException e)
183.         {
184.             throw new RuntimeException(e);
185.         }
186.     }
187. }
```

10.1.5.2. Package-info.java

```
1. /**
2.  * Dieses Paket enthält die Main-Klasse
3.  * @author Philipp Hennken
4.  * @version 18.0.2
5.  */
6. package view;
```

10.2. Javadoc Dokumentation

Contents

Class Hierarchy	3
1 Package control	4
1.1 Class DatenManager	4
1.1.1 Declaration	4
1.1.2 Constructor summary	4
1.1.3 Method summary	4
1.1.4 Constructors	5
1.1.5 Methods	5
1.2 Class ZeitMessung	6
1.2.1 Declaration	7
1.2.2 Constructor summary	7
1.2.3 Method summary	7
1.2.4 Constructors	7
1.2.5 Methods	7
2 Package model	8
2.1 Class Graph	8
2.1.1 Declaration	8
2.1.2 Constructor summary	8
2.1.3 Method summary	8
2.1.4 Constructors	9
2.1.5 Methods	9
2.2 Class LadeStation	10
2.2.1 Declaration	10
2.2.2 Constructor summary	10
2.2.3 Method summary	10
2.2.4 Constructors	11
2.2.5 Methods	11
2.3 Class WegFinder	12
2.3.1 Declaration	12
2.3.2 Constructor summary	12
2.3.3 Method summary	12
2.3.4 Constructors	12
2.3.5 Methods	12
2.4 Exception LadestationNichtInGraphException	13

2.4.1	Declaration	13
2.4.2	Constructor summary	13
2.4.3	Constructors	13
2.4.4	Members inherited from class <code>Throwable</code>	13
3	Package <code>res</code>	14
3.1	Interface <code>MeineKonstante</code>	14
3.1.1	Declaration	14
3.1.2	Field summary	14
3.1.3	Fields	15
3.2	Interface <code>Strings</code>	15
3.2.1	Declaration	15
3.2.2	Field summary	15
3.2.3	Fields	16
4	Package <code>util</code>	18
4.1	Class <code>MyUtil</code>	18
4.1.1	Declaration	18
4.1.2	Constructor summary	18
4.1.3	Method summary	18
4.1.4	Constructors	19
4.1.5	Methods	19
5	Package <code>view</code>	21
5.1	Class <code>Main</code>	21
5.1.1	Declaration	21
5.1.2	Constructor summary	21
5.1.3	Method summary	21
5.1.4	Constructors	21
5.1.5	Methods	22

Class Hierarchy

Classes

- `java.lang.Object`
 - `control.DatenManager` (in [1.1](#), page [4](#))
 - `control.ZeitMessung` (in [1.2](#), page [6](#))
 - `model.Graph` (in [2.1](#), page [8](#))
 - `model.LadeStation` (in [2.2](#), page [10](#))
 - `model.WegFinder` (in [2.3](#), page [12](#))
 - `util.MyUtil` (in [4.1](#), page [18](#))
 - `view.Main` (in [5.1](#), page [21](#))

Interfaces

- `res.MeineKonstante` (in [3.1](#), page [14](#))
- `res.Strings` (in [3.2](#), page [15](#))

Exceptions

- `java.lang.Object`
- `java.lang.Throwable`
- `java.lang.Exception`
- `model.LadestationNichtInGraphException` (in [2.4](#), page [13](#))

Chapter 1

Package control

<i>Package Contents</i>	<i>Page</i>
Classes	
DatenManager	4
In dieser Klasse können Daten eingelesen, sortiert und Graphen erstellt sowie ausgewertet werden.	
ZeitMessung	6
In dieser Klasse wird die Zeitmessung der einzelnen Methoden gesteuert.	

Dieses Paket enthält Klassen zum Verarbeiten von Daten.

1.1 Class DatenManager

In dieser Klasse können Daten eingelesen, sortiert und Graphen erstellt sowie ausgewertet werden.

1.1.1 Declaration

```
public class DatenManager
    extends java.lang.Object
```

1.1.2 Constructor summary

[DatenManager\(\)](#)

1.1.3 Method summary

[erhalteLadeStationVonPostleitzahl\(int, Graph\)](#) Falls vorhanden, gibt diese Methode ein Objekt vom Typ "LadeStation" mit der Postleitzahl zurück, die ihr übergeben wird.
[getAnzahlEitraege\(\)](#)

leseDatenZeilenweise(BufferedReader) Diese Methode fügt Objekte der Klasse "LadeStation" zu einer ArrayList hinzu und löscht anschließend alle Ladestationen heraus, deren Geo-Koordinaten nicht möglich sind.

loescheEintraegeInEpsilonUmgebung(ArrayList, int) Diese Methode löscht Objekte des Typs "LadeStation" in der Epsilonumgebung einer Referenzstation, die vom User eingegeben wird.

sortieren(ArrayList) Diese Methode sortiert eine ArrayList mit Objekten des Typs "LadeStation" zuerst aufsteigend nach der Postleitzahl.

1.1.4 Constructors

- **DatenManager**

```
public DatenManager()
```

1.1.5 Methods

- **erhalteLadeStationVonPostleitzahl**

```
public static model.LadeStation
    erhalteLadeStationVonPostleitzahl(int postleitzahl, model.
    Graph graph)
```

- **Description**

Falls vorhanden, gibt diese Methode ein Objekt vom Typ "LadeStation" mit der Postleitzahl zurück, die ihr übergeben wird.

- **Parameters**

- * **postleitzahl** – Die Postleitzahl, die das Objekt vom Typ "LadeStation" haben soll.
- * **graph** – Der Graph, in dem das Objekt vom Typ "LadeStation" gesucht werden soll.

- **Returns** – Falls vorhanden, das Objekt vom Typ "LadeStation" mit der eingegebenen Postleitzahl, sonst null.

- **getAnzahlEitraege**

```
public int getAnzahlEitraege()
```

- **leseDatenZeilenweise**

```
public java.util.ArrayList leseDatenZeilenweise(java.io.
    BufferedReader reader)
```


- **Description**

Diese Methode fügt Objekte der Klasse "LadeStation" zu einer ArrayList hinzu und löscht anschließend alle Ladestationen heraus, deren Geo-Koordinaten nicht möglich sind. Zusätzlich gibt die Methode die Bearbeitungszeit, sowie die Anzahl der eingelesenen Dateien aus.

- **Parameters**

- * **reader** – BufferedReader der Daten aus einer Datei Zeilenweise einliest.

- **Returns** – ArrayList mit Objekten des Typs "LadeStation" und korrekten Geo-Koordinaten

- **loescheEintraegeInEpsilonUmgebung**

```
public java.util.ArrayList loescheEintraegeInEpsilonUmgebung(
    java.util.ArrayList sortierteListe ,int epsilonUmgebung)
```

- **Description**

Diese Methode löscht Objekte des Typs "LadeStation" in der Epsilonumgebung einer Referenzstation, die vom User eingegeben wird. Eine Vergleichsstation, die sich innerhalb der Epsilonumgebung der Referenzstation befindet, wird gelöscht.

- **Parameters**

- * **sortierteListe** – Eine ArrayList die Objekte des Typs "LadeStation" enthält und nach der Postleitzahl aufsteigend sortiert ist.

- * **epsilonUmgebung** – Die Entfernung zwischen zwei Ladesäulen, die mindestens bestehen muss, damit die Vergleichsstation nicht gelöscht wird.

- **Returns** – Eien ArrayList, die sortiert ist und nur Objekte vom Typ "LadeStation" enthält, die mindestens eine Epsilonumgebung voneinander entfernt sind.

- **sortieren**

```
public java.util.ArrayList sortieren(java.util.ArrayList
    arrayListZuSortieren)
```

- **Description**

Diese Methode sortiert eine ArrayList mit Objekten des Typs "LadeStation" zuerst aufsteigend nach der Postleitzahl. Bei zwei gleichen Postleitzahlen wird absteigend nach der Anschlussleitung einer Ladesäule sortiert.

- **Parameters**

- * **arrayListZuSortieren** – Die ArrayList, die sortiert werden soll.

- **Returns** – Die fertig sortierte ArrayList.

1.2 Class ZeitMessung

In dieser Klasse wird die Zeitmessung der einzelnen Methoden gesteuert.

1.2.1 Declaration

```
public class ZeitMessung
    extends java.lang.Object
```

1.2.2 Constructor summary

[ZeitMessung\(\)](#)

1.2.3 Method summary

[berechneVergangeneZeitInMillisekunden\(\)](#) Diese Methode berechnet die Zeit, die seit dem Start der Zeitmessung vergangen ist.
[start\(\)](#) Diese Methode startet die Zeitmessung.

1.2.4 Constructors

- **ZeitMessung**

```
public ZeitMessung()
```

1.2.5 Methods

- **berechneVergangeneZeitInMillisekunden**

```
public static long berechneVergangeneZeitInMillisekunden()
```

- **Description**

Diese Methode berechnet die Zeit, die seit dem Start der Zeitmessung vergangen ist. Dafür wird die Differenz der jetzigen Zeit und der Zeit zum Startzeitpunkt genutzt. Die Zahl wird durch 1000 dividiert, damit die Zeit in Millisekunden angegeben werden kann.

- **Returns** – Die Zeit die seit dem Start der Zeitmessung vergangen ist in Millisekunden.

- **start**

```
public static void start()
```

- **Description**

Diese Methode startet die Zeitmessung. Dafür wird die aktuelle Zeit in Nanosekunden als Startzeit gesetzt.

Chapter 2

Package model

<i>Package Contents</i>	<i>Page</i>
Classes	
Graph	8
In dieser Klasse wird ein Graph erstellt und verwaltet.	
LadeStation	10
In dieser Klasse werden alle relevanten Daten einer Ladestation abgespeichert und verwendbar gemacht.	
WegFinder	12
In dieser Klasse wird nach einem möglichen Weg zwischen zwei Objekten des Typs "LadeStation" gesucht.	
Dieses Paket modelliert Datenstrukturen.	

2.1 Class Graph

In dieser Klasse wird ein Graph erstellt und verwaltet.

2.1.1 Declaration

```
public class Graph
    extends java.lang.Object
```

2.1.2 Constructor summary

Graph(ArrayList) Dieser Konstruktor weist die übergebene ArrayList der internen zu.

2.1.3 Method summary

erzeugeGarphen(int) Diese Methode erzeugt einen Graphen mit einem Objekt des Typs "LadeStation" als Key und einer ArrayList mit Objekten des Typs "LadeStation" als Value.

getGraph()

getGraphKeys() Diese Methode gibt alle Keys eines Graphen wieder.

getStringOf(HashMap) Diese Methode gibt einen String für eine schöne Ausgabe einer HashMap zurück.

setKnoten(HashMap)

2.1.4 Constructors

- **Graph**

```
public Graph(java.util.ArrayList arrayListMitLadeStationen)
```

- **Description**

Dieser Konstruktor weist die übergebene ArrayList der internen zu.

- **Parameters**

* **arrayListMitLadeStationen** – Die übergebene ArrayList mit Objekten vom Typ "LadeStation".

2.1.5 Methods

- **erzeugeGarphen**

```
public java.util.HashMap erzeugeGarphen(int reichweiteAuto)
```

- **Description**

Diese Methode erzeugt einen Graphen mit einem Objekt des Typs "LadeStation" als Key und einer ArrayList mit Objekten des Typs "LadeStation" als Value. Die Epsilonumgebung beschreibt den Mindestabstand zwischen zwei Ladesäulen, sodass sie gelöscht werden, wenn sie zu nah aneinander stehen.

- **Parameters**

* **reichweiteAuto** – Der Mindestabstand, der zwischen zwei Ladesäulen sein muss, damit eine der beiden nicht gelöscht wird.

- **Returns** – Eine HashMap mit einem Objekt des Typs "LadeStation" als Key und einer ArrayList mit Objekten des Typs "LadeStation" als Value.

- **getGraph**

```
public java.util.HashMap getGraph()
```

- **getGraphKeys**

```
public java.util.Set getGraphKeys()
```

- **Description**
Diese Methode gibt alle Keys eines Graphen wieder.
- **Returns** – Alle Keys des Graphen.

- **getStringOf**

```
public static java.lang.String getStringOf(java.util.HashMap
    ladeStationArrayListHashMap)
```

- **Description**
Diese Methode gibt einen String für eine schöne Ausgabe einer HashMap zurück.
- **Parameters**
* ladeStationArrayListHashMap – Die HashMap die ausgegeben werden soll.
- **Returns** – Ein String der eine HashMap mit einem Objekt des Typs "LadeStation" als Key und einer ArrayList mit Objekten des Typs "LadeStation" als Value schön anzeigt.

- **setKnoten**

```
public void setKnoten(java.util.HashMap knoten)
```

2.2 Class LadeStation

In dieser Klasse werden alle relevanten Daten einer Ladestation abgespeichert und verwendbar gemacht.

2.2.1 Declaration

```
public class LadeStation
    extends java.lang.Object
```

2.2.2 Constructor summary

LadeStation(String) In diesem Konstruktor werden Objekte vom Typ "LadeStation" erstellt und die relevanten Attribute zugewiesen.

2.2.3 Method summary

```
getAnschlussleistung()
getBetreiber()
getBreitengrad()
getLaengengrad()
getOrt()
```

```
getPostleitzahl()  
getStrasse()  
toString()
```

2.2.4 Constructors

- **LadeStation**

```
public LadeStation(java.lang.String input)
```

- **Description**

In diesem Konstruktor werden Objekte vom Typ "LadeStation" erstellt und die relevanten Attribute zugewiesen.

- **Parameters**

- * **input** – Eine eingelesene Zeile der auszuwertenden CSV-Datei, die durch Simikola getrennt sind.

2.2.5 Methods

- **getAnschlussleistung**

```
public java.lang.Float getAnschlussleistung()
```

- **getBetreiber**

```
public java.lang.String getBetreiber()
```

- **getBreitengrad**

```
public float getBreitengrad()
```

- **getLaengengrad**

```
public float getLaengengrad()
```

- **getOrt**

```
public java.lang.String getOrt()
```

- **getPostleitzahl**

```
public int getPostleitzahl()
```

- `getStrasse`

```
public java.lang.String getStrasse()
```

- `toString`

```
public java.lang.String toString()
```

2.3 Class WegFinder

In dieser Klasse wird nach einem möglichen Weg zwischen zwei Objekten des Typs "LadeStation" gesucht.

2.3.1 Declaration

```
public class WegFinder
    extends java.lang.Object
```

2.3.2 Constructor summary

[WegFinder\(\)](#)

2.3.3 Method summary

[findeWeg\(Graph, int, int, ArrayList\)](#) Diese Methode findet einen Weg von einer Ladestation in dem Ort, mit der ersten eingegebenen Postleitzahl zu einer Ladestation in dem Ort mit der anderen eingegebenen Postleitzahl.

2.3.4 Constructors

- `WegFinder`

```
public WegFinder()
```

2.3.5 Methods

- `findeWeg`

```
public void findeWeg(Graph graph, int startPLZ, int endPLZ, java.
    util.ArrayList besuchteLadeStationen) throws model.
    LadestationNichtInGraphException
```

- **Description**

Diese Methode findet einen Weg von einer Ladestation in dem Ort, mit der ersten eingegebenen Postleitzahl zu einer Ladestation in dem Ort mit der anderen eingegebenen Postleitzahl. Die Suche des Weges basiert auf einer Tiefensuche.

- **Parameters**

- * **graph** – In diesem Graphen wird nach den Haltestellen für den Weg zwischen den Orten gesucht.
- * **startPLZ** – Die Postleitzahl, von dem Ort, wo der Weg beginnen soll.
- * **endPLZ** – Die Postleitzahl, von dem Ort, wo der Weg enden soll.

2.4 Exception LadestationNichtInGraphException

In dieser Klasse wird eine Exception formuliert.

2.4.1 Declaration

```
public class LadestationNichtInGraphException
    extends java.lang.Exception
```

2.4.2 Constructor summary

[LadestationNichtInGraphException\(\)](#) Der Konstruktor ruft den Konstruktor der Superklasse "Exception" auf.

2.4.3 Constructors

- **LadestationNichtInGraphException**

```
public LadestationNichtInGraphException()
```

- **Description**

Der Konstruktor ruft den Konstruktor der Superklasse "Exception" auf.

2.4.4 Members inherited from class Throwable

```
java.lang.Throwable
• public final synchronized void addSuppressed(Throwable arg0)
• public synchronized Throwable fillInStackTrace()
• public synchronized Throwable getCause()
• public String getLocalizedMessage()
• public String getMessage()
• public StackTraceElement getStackTrace()
• public final synchronized Throwable getSuppressed()
• public synchronized Throwable initCause(Throwable arg0)
• public void printStackTrace()
• public void printStackTrace(java.io.PrintStream arg0)
• public void printStackTrace(java.io.PrintWriter arg0)
• public void setStackTrace(StackTraceElement[] arg0)
• public String toString()
```


Chapter 3

Package res

<i>Package Contents</i>	<i>Page</i>
Interfaces	
MeineKonstante	14
In diesem Interface befinden sich Konstanten, damit der Code übersichtlich bleibt.	
Strings	15
In diesem Interface befinden sich Literale, damit der Code übersichtlich bleibt.	

Dieses Paket enthält programmspezifische Daten.

3.1 Interface MeineKonstante

In diesem Interface befinden sich Konstanten, damit der Code übersichtlich bleibt.

3.1.1 Declaration

```
public interface MeineKonstante
```

3.1.2 Field summary

```
ANSCHLUSSLEISTUNG_INDEX  
BETREIBER_INDEX  
BREITENGRAD_INDEX  
BUNDESLAND_INDEX  
HAUSNUMMER_INDEX  
LAENGENGRAD_INDEX  
NANO_TO_MILLI  
ORT_INDEX  
POSTLEITZAHL_INDEX  
STRASSE_INDEX  
ZERO
```

3.1.3 Fields

- `int NANO_TO_MILLI`
- `int BETREIBER_INDEX`
- `int STRASSE_INDEX`
- `int HAUSNUMMER_INDEX`
- `int POSTLEITZAHL_INDEX`
- `int ORT_INDEX`
- `int BUNDESLAND_INDEX`
- `int BREITENGRAD_INDEX`
- `int LAENGENGRAD_INDEX`
- `int ANSCHLUSSLEISTUNG_INDEX`
- `int ZERO`

3.2 Interface Strings

In diesem Interface befinden sich Literale, damit der Code übersichtlich bleibt.

3.2.1 Declaration

```
public interface Strings
```

3.2.2 Field summary

```
ABSATZ  
ANSCHLUSSLEISTUNG  
AUSGABE_LISTE_WURDE_SORTIERT  
DOPPELPUNKT  
EINHEIT_ANACHLUSSLEISTUNG  
EINHEIT_ZEITMESSUNG  
EINLESEN_ZEIT  
END_PLZ_FRAGE  
EPSILONUMGEBUNG_ABFRAGE  
EPSILONUMGEBUNG_FALSCH  
ES_SIND  
FALSCHES_PLZ_EINGABE  
FILE_NAME  
FILE_NAME_ZUM_TESTEN  
GRAPH_ERSTELLEN
```

GUELTIGE_PLZ_SIND
LADESTATION_POST_ANBIETER
LADESTATION_PRE_ANBIETER
LADESTATIONEN_EINGETRAGEN
LEERZEICHEN
LISTE_ALLER_VERBINDUNGEN
LOESCHEN_ZEIT
PFEIL
PLZ
REICHWEITE_FALSCH
REICHWEITE_FRAGE
SIMIKOLON
SORTIEREN_ZEIT
START_PLZ_FRAGE
WEG_FINDER
WEG_NICHT_VORHANDEN_POST
WEG_VORHANDEN_NACH
WEG_VORHANDEN_POST
WEG_VORHANDEN_PRE

3.2.3 Fields

- java.lang.String SIMIKOLON
- java.lang.String EINHEIT_ZEITMESSUNG
- java.lang.String EINHEIT_ANACHLUSSLEISTUNG
- java.lang.String LEERZEICHEN
- java.lang.String LADESTATION_PRE_ANBIETER
- java.lang.String LADESTATION_POST_ANBIETER
- java.lang.String FILE_NAME
- java.lang.String FILE_NAME_ZUM_TESTEN
- java.lang.String START_PLZ_FRAGE
- java.lang.String END_PLZ_FRAGE
- java.lang.String FALSCHHE_PLZ_EINGABE
- java.lang.String GUELTIGE_PLZ_SIND
- java.lang.String REICHWEITE_FALSCH
- java.lang.String REICHWEITE_FRAGE
- java.lang.String EPSILONUMGEBUNG_FALSCH

- `java.lang.String EPSILONUMGEBUNG_ABFRAGE`
- `java.lang.String LISTE_ALLER_VERBINDUNGEN`
- `java.lang.String PFEIL`
- `java.lang.String ABSATZ`
- `java.lang.String WEG_VORHANDEN_PRE`
- `java.lang.String DOPPELPUNKT`
- `java.lang.String PLZ`
- `java.lang.String ANSCHLUSSLEISTUNG`
- `java.lang.String SORTIEREN_ZEIT`
- `java.lang.String EINLESEN_ZEIT`
- `java.lang.String LOESCHEN_ZEIT`
- `java.lang.String GRAPH_ERSTELLEN`
- `java.lang.String WEG_FINDER`
- `java.lang.String WEG_VORHANDEN_NACH`
- `java.lang.String WEG_VORHANDEN_POST`
- `java.lang.String WEG_NICHT_VORHANDEN_POST`
- `java.lang.String ES_SIND`
- `java.lang.String LADESTATIONEN_EINGETRAGEN`
- `java.lang.String AUSGABE_LISTE_WURDE_SORTIERT`

Chapter 4

Package util

<i>Package Contents</i>	<i>Page</i>
Classes	
MyUtil	18
In dieser Klasse werden Ein- und Ausgaben auf der Konsole gesteuert und verwaltet.	

Dieses Paket enthält Klassen zum Ein- und Ausgeben.

4.1 Class MyUtil

In dieser Klasse werden Ein- und Ausgaben auf der Konsole gesteuert und verwaltet.

4.1.1 Declaration

```
public class MyUtil
    extends java.lang.Object
```

4.1.2 Constructor summary

[MyUtil\(\)](#)

4.1.3 Method summary

[fehlerAusgeben\(Object\)](#) Diese Methode gibt ein Objekt als Fehlermeldung aus.

[gebeAnbieterUndPlzUndAnschlussleistungAus\(ArrayList\)](#) Diese Methode gibt den Anbieter, die Postleitzahl und die Anschlussleistung einer ArrayList von Ladestationen aus.

[haversine\(double, double, double, double\)](#) Diese Methode ermittelt den Abstand zwischen zwei Punkten auf einer Kugel anhand von Geokoordinaten.

[intEingabe\(\)](#) Diese Methode fordert vom Benutzer eine Eingabe einer ganzen Zahl und gibt diese wieder.

[normalAusgeben\(Object\)](#) Diese Methode gibt ein Objekt aus.

4.1.4 Constructors

- **MyUtil**

```
public MyUtil()
```

4.1.5 Methods

- **fehlerAusgeben**

```
public static void fehlerAusgeben(java.lang.Object ausgabeObjekt  
)
```

- **Description**

Diese Methode gibt ein Objekt als Fehlermeldung aus.

- **Parameters**

- * `ausgabeObjekt` – Das übergebene Objekt, das als Fehlermeldung ausgegeben werden soll.

- **gebeAnbieterUndPlzUndAnschlussleitungAus**

```
public static void gebeAnbieterUndPlzUndAnschlussleitungAus(java  
.util.ArrayList daten)
```

- **Description**

Diese Methode gibt den Anbieter, die Postleitzahl und die Anschlussleistung einer ArrayList von Ladestationen aus.

- **Parameters**

- * `daten` – ArrayList mit Ladestationen.

- **haversine**

```
public static double haversine(double BreitengradEins, double  
LaengengradEins, double BreitengradZwei, double LaengengradZwei  
)
```

- **Description**

Diese Methode ermittelt den Abstand zwischen zwei Punkten auf einer Kugel anhand von Geokoordinaten. Die Methode stammt von der Internetseite <https://de.acervolima.com/haversine-formel-zum-ermitteln-des-abstands-zwischen-zwei-punkten-auf-einer-kugel/>.

- **Parameters**

- * `BreitengradEins` – Breitengrad des ersten Ortes.

- * `LaengengradEins` – Längengrad des ersten Ortes.
- * `BreitengradZwei` – Breitengrad des zweiten Ortes.
- * `LaengengradZwei` – Längengrad des ersten Ortes.
- **Returns** – Abstand zwischen den beiden Orten in Kilometern.

- **intEingabe**

```
public static int intEingabe() throws java.io.IOException
```

- **Description**
Diese Methode fordert vom Benutzer eine Eingabe einer ganzen Zahl und gibt diese wieder.
- **Returns** – Der eingegebene Zahlenwert des Benutzers.
- **Throws**
 - * `java.io.IOException` –

- **normalAusgeben**

```
public static void normalAusgeben(java.lang.Object ausgabeObjekt  
)
```

- **Description**
Diese Methode gibt ein Objekt aus.
- **Parameters**
 - * `ausgabeObjekt` – Das übergebene Objekt, das ausgegeben werden soll.

Chapter 5

Package view

<i>Package Contents</i>	<i>Page</i>
Classes	
Main	21
In dieser Klasse wird das Programm gestartet.	

Dieses Paket enthält die Main-Klasse

5.1 Class Main

In dieser Klasse wird das Programm gestartet.

5.1.1 Declaration

```
public class Main
    extends java.lang.Object
```

5.1.2 Constructor summary

[Main\(\)](#)

5.1.3 Method summary

[main\(String\[\]\)](#) Diese Methode wird zum Starten des Programms genutzt.

5.1.4 Constructors

- Main

```
public Main()
```


5.1.5 Methods

- **main**

```
public static void main(java.lang.String[] args) throws java.io.  
IOException, model.LadestationNichtInGraphException
```

- **Description**

Diese Methode wird zum Starten des Programms genutzt.

- **Parameters**

- * **args** – Die Programmargumente.

- **Throws**

- * **java.io.IOException** – Eine IOException kann auftreten, wenn falsche Daten vom Nutzer eingegeben werden.
- * **model.LadestationNichtInGraphException** – Sollte eine Postleitzahl eingegeben werden, die keiner Ladestation im Graphen zugeordnet ist, wird diese Exception geworfen.