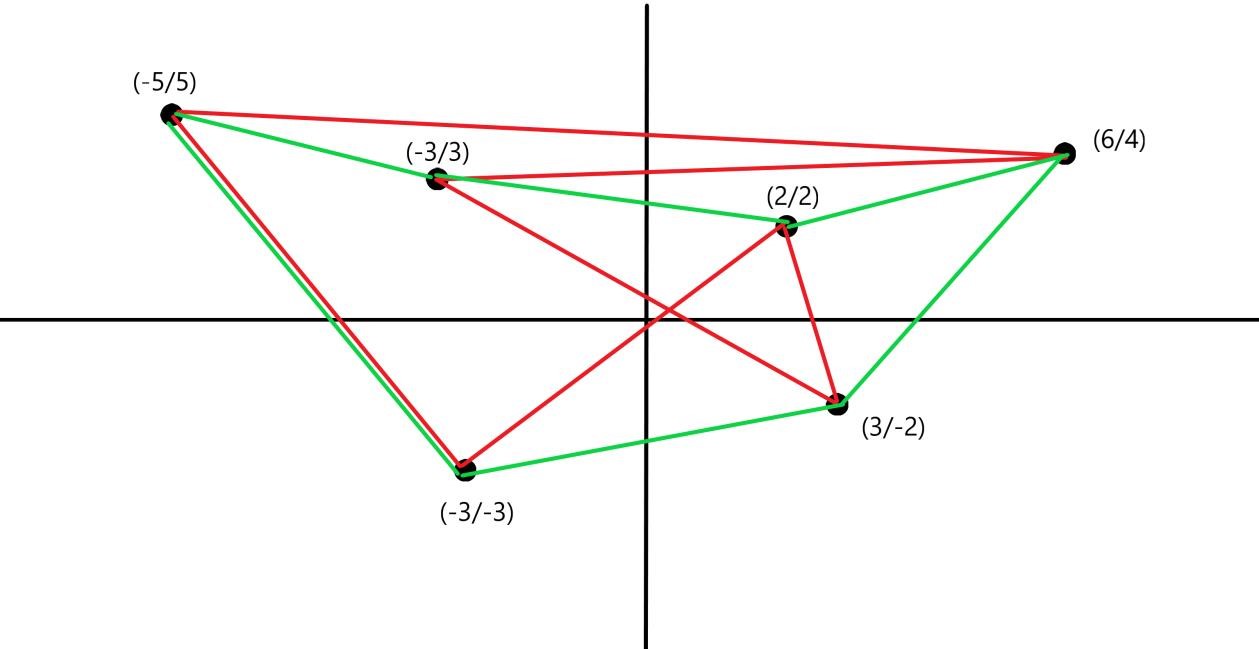
**Quickest Travel Route Calculator:**

Unsere Aufgabenstellung ist wie folgt:

Aus einem File werden die Koordinaten eines Startpunkts sowie von einer Serie von zu besuchenden

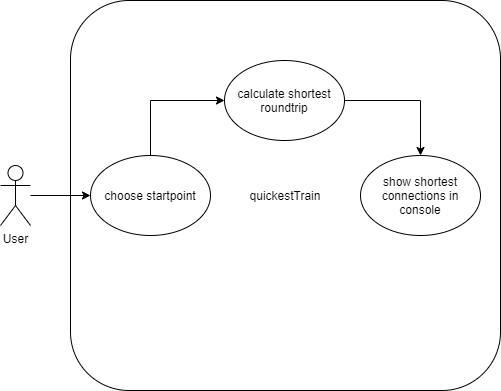
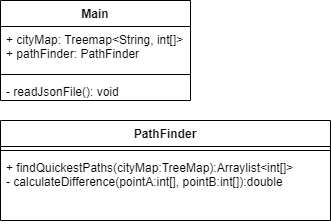
Orten eingelesen. Die Applikation soll dann den kürzesten Weg finden, der vom Startpunkt zu allen Orten und wieder zurück zum Startpunkt führt. Als Distanz jedes Streckenteils wird die Luftlinie zwischen zwei Orten verwendet.

Die beste Reiseroute sollte auf der Konsole und in einem Textfile ausgegeben werden (Liste in der Reihenfolge der besuchten Orte mit Distanz).



**Analyse**

Aus dieser Analyze heraus entstanden grobe Klassendiagramme und ein Use Case Diagram:



Funktionen:

* Koordinaten via einem Json File lesen
* Schnellsten Weg finden
* Koordinaten Reihenfolge in einem File speichern

Mit den Anforderungen wird eine simple Konsolen Applikation reichen. Sobald ein Startpunkt gewählt ist, passiert der Rest im Hintergrund. Unten ein kleines Mockup des Menus in Text-Form

/\*\*\*/

[0] Bern

[1] Genf

[2] Luzern

[3] St.Gallen

[4] Zürich

Choose your startpoint:4

/\*\*\*/

You chose Zürich

Zürich[1, 2]

Bern[1, 3]

Genf[2, 4]

St.Gallen[9, 2]

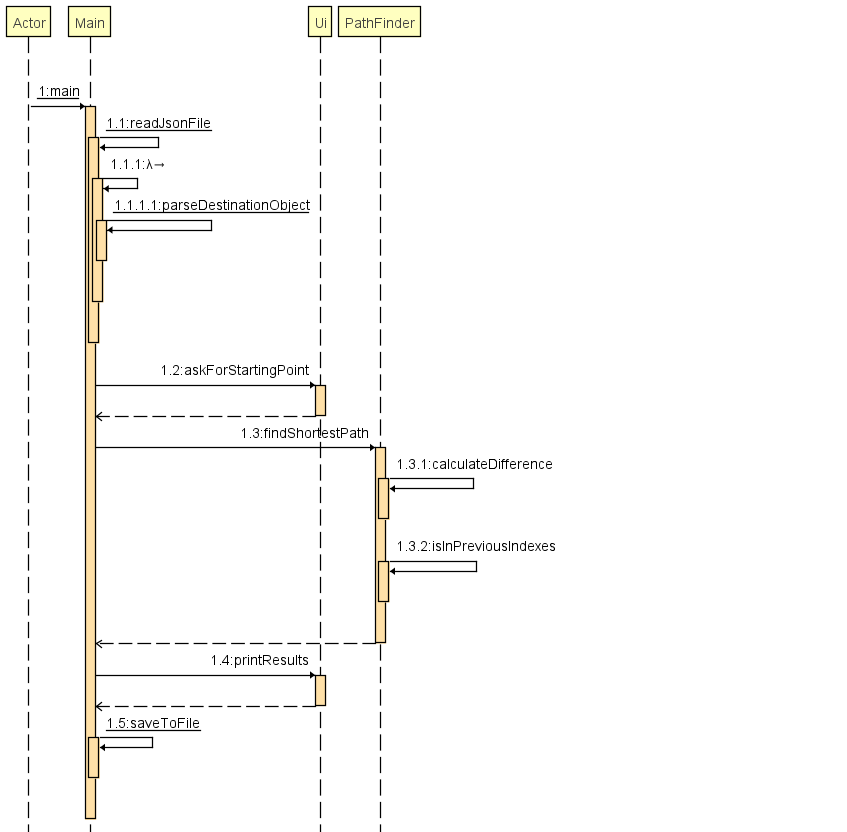
Luzern[10, 3]

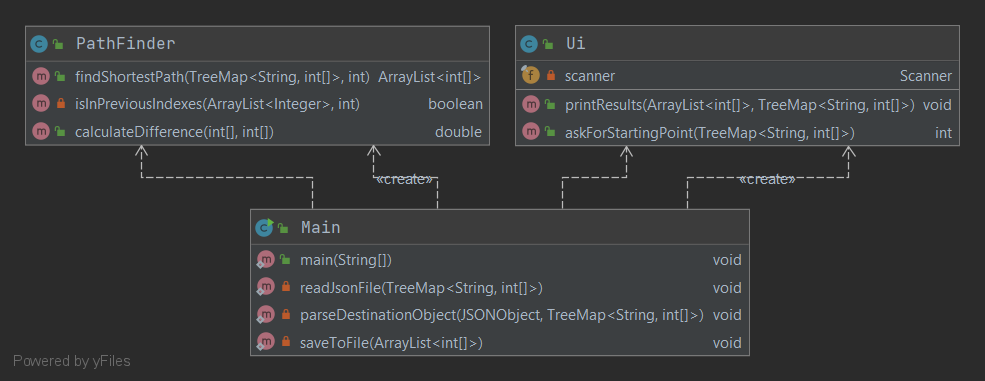
Zürich[1, 2]

/\*\*\*/

**Design**

Im Design kommt eine zusätzlichen Ui Klasse für die Konsolen Ausgaben und einer Methode um die Reihenfolge der Koordinaten in einem separaten File zu speichern.

Unten die finale Klassendiagram für die Implementierung und das Sequenzdiagram:



**Implementierung:**

In der Implenetierungsphase kostete der PathFinder Algorithmus am meisten Zeit.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Die Konsolenapplikation sieht sehr ähnlich wie unser kleines Text-Mockup aus und der Algorithmus funktioniert nach vielem Testing.

Algorithmus Simplified:

Das Programm erhält eine List von Koordinaten Sets. Vom Benutzer definierten Startpunkt aus startet die «Reise». Der allererste Startpunkt wird direkt als besucht gekennzeichnet und kommt in die Liste des kürzesten Wegs (die Liste des kürzesten Wegs wird am Schluss zurückgegeben). Das Programm vergleicht die Distanz vom Startpunkt zu allen Punkten und nehmt den kürzesten Weg. Angenommen vom Startpunkt (A) aus liegt Punkt (C) am nächsten. Dann wird Punkt (C) zum nächsten Startpunkt und zu dem bereits besuchten Punkt gekennzeichnet. Von Punkt(C) aus vergleicht das Programm wieder alle Distanzen. Von Punkt(C) aus liegt Punkt(F) am nächsten. Punkt(F) wird zum nächsten Start Punkt und zu dem bereits besuchten Punkt gekennzeichnet usw., bis die Reise wieder beim Startpunkt ist.

**Testprotokoll "Travaling Salesman"**

Autor : Raphael Blaauw, Oliver Achermann

Geprüfte SW-Version: 1.0

# **Testgruppe 1: Resultaten werden getestet**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Testfall | Vorgegeben Liste | Index | Erwartetes Resultat | Tatsächliches Resultat | OK/NOK |
| 01 | {{8,8},{1,1},{3,3}} | 0 | 1 | 1 | OK |
| 02 | {{8,8},{1,1},{3,3}} | 1 | 3 | 3 | OK |
| 03 | {{8,8},{1,1},{3,3}} | 2 | 8 | 8 | OK |
| 04 | {{8,8},{1,1},{3,3}} | 3 | 1 | 1 | OK |

# **Testgruppe 2: Distanz Kalkulator wird getestet**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Testfall** | **Punkt 1** | **Punkt 2** | **Erwartetes Resultat** | **Tatsächliches Resultat** | **OK/NOK** | |
| **01** | **1** | **1** | 1.414213562 | 1.414213562 | **OK** |

**Der Prüfer**

Oliver Achermann, 09.11.2021, Zürich

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

@Before  
public void init(){  
 *cityMap* = new TreeMap<>()**;** *cityMap*.put("Zürich"**,** new int[]{8**,** 8})**;** *cityMap*.put("Argau"**,** new int[]{1**,** 1})**;** *cityMap*.put("Luzern"**,** new int[]{3**,** 3})**;** *pf* = new PathFinder()**;**}  
  
@Test  
public void testShortestPathStartingPoint(){  
 int[] result = *pf*.findShortestPath(*cityMap***,** 0).get(0)**;** Assert.*assertEquals*(1**,**result[0])**;**}  
@Test  
public void testShortestPathSecondPoint(){  
 int[] result = *pf*.findShortestPath(*cityMap***,** 0).get(1)**;** Assert.*assertEquals*(3**,**result[0])**;**}  
  
@Test  
public void testShortestPathThirdPoint(){  
 int[] result = *pf*.findShortestPath(*cityMap***,** 0).get(2)**;** Assert.*assertEquals*(8**,**result[0])**;**}  
  
@Test  
public void testShortestPathEndPoint(){  
 int[] result = *pf*.findShortestPath(*cityMap***,** 0).get(3)**;** Assert.*assertEquals*(1**,**result[0])**;**}  
  
@Test  
public void calculateDistanceTest(){  
 PathFinder pf = new PathFinder()**;** double diff = pf.calculateDifference(new int[]{0**,** 0}**,**new int[]{1**,** 1})**;** Assert.*assertEquals*(1.4142135623730951**,**diff)**;**}

**Reflektionen**

**Raphael Blaauw**

Das Mini-Projekt war eine gute Abrundung für das Modul 411. Nochmals selber einen Algorithmus zu entwerfen machte Spass war aber auch eine Herausforderung. Ich lernte, wie man ein Json-File in ein Java-Projekt parsen konnte, und setzte dies gleich in einem anderen Projekt ein. Ebenfalls war es spannend den Pathfinder Algorithmus so anzupassen das er auch andere Startpunkte entgegennehmen kann.

**Oliver Achermann**

Ich habe mich gefreut, selbst programmierten Algorithmen in einem Projekt integrieren zu können. Dabei hatte ich Spaß, aber auch Mühe. Gelernt habe ich, was ein JSON überhaupt ist und wie man sie parsed und verwendet. Allgemein im Modul konnte ich vieles Lernen bezüglich Algorithmen. Bereits vorher habe ich von Linked List gehört, jedoch nie damit auseinandergesetzt. Bubble sort habe ich gekannt, konnte es jedoch nicht nennen. Das gleiche gilt auch für linear search. Völlig neu für mich waren quicksearch und binarysearch.