Main.java:



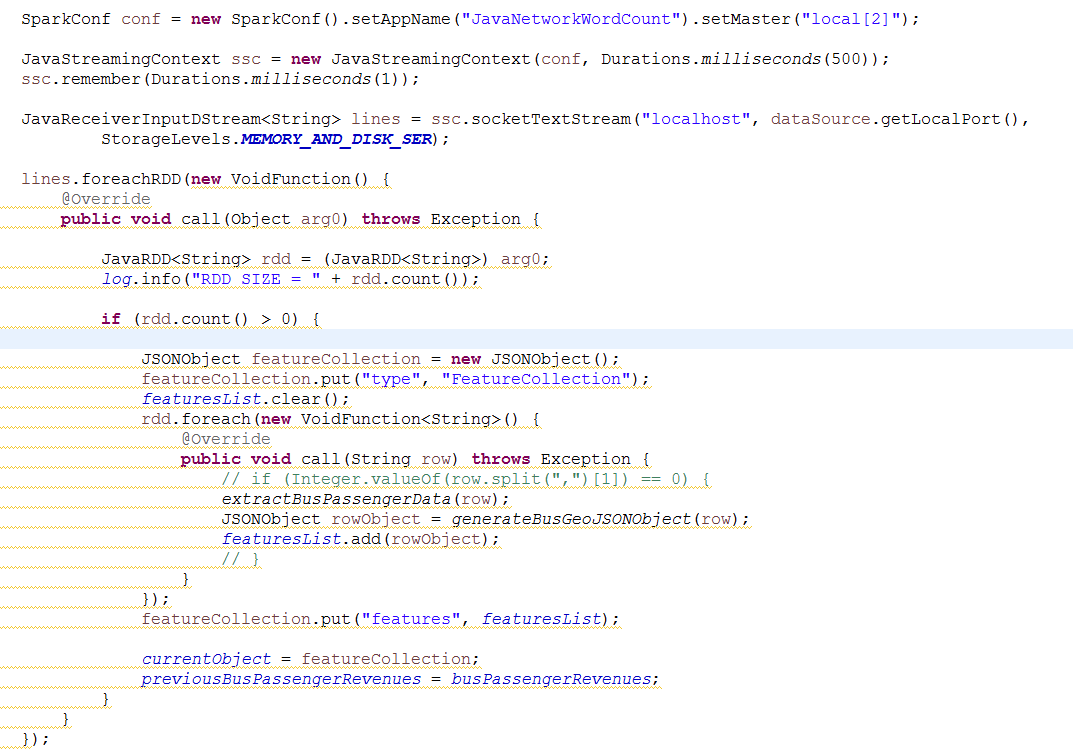
Setzt den Spark Webserver auf und definiert die URLs der GET Anfragen. /data liefert die JSON repräsentation der Busse für die Karte, /tsv die revenues pro passagier (bus egal) und /buses liefert die gesamte revenue aller passagiere in einem speziellen bus

Zusätzlich wird über die Methode im Try-catch block die Webseite automatisch aufgerufen, sodass der Nutzer dies nicht mehr extra machen muss

readCSV-Methode: neben setupWebServer die wichtigste Methode – handelt alles was csv/json lesen/generieren anbelangt ab

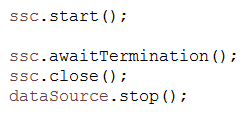


Hier wird die csv-Datei eingelesen (BufferedReader liest dabei je eine Zeile ein); ServerSocketSource (aus git-example) fungiert hier mehr oder weniger als „Stream“ Produzent und liest die csv Datei nach und nach mit Hilfe des BufferedReaders ein. 🡪 Genauer: Es wird so lange eingelesen, wie gleicher Timestamp gefunden wird. Wird unterschiedlicher Timestamp gefunden, wird der Thread 100ms lang pausiert. Dann wird mit dem nächsten Timestamp wieder zu lesen begonnen (solange bis dass wieder nächsthöherer gefunden wird)



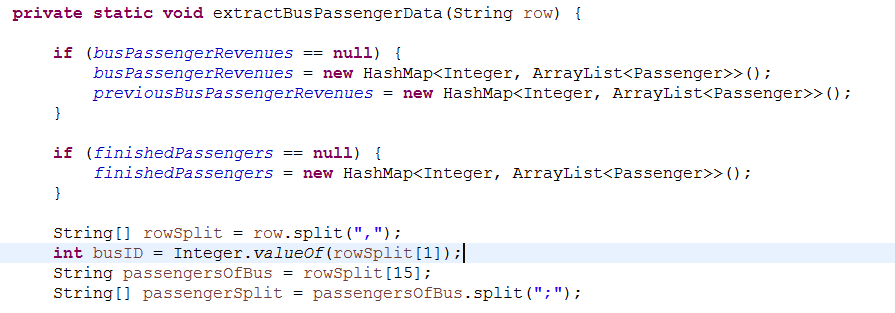
Hier wird dann eigentlicher Stream-Consumer über Spark Streaming Context erstellt.

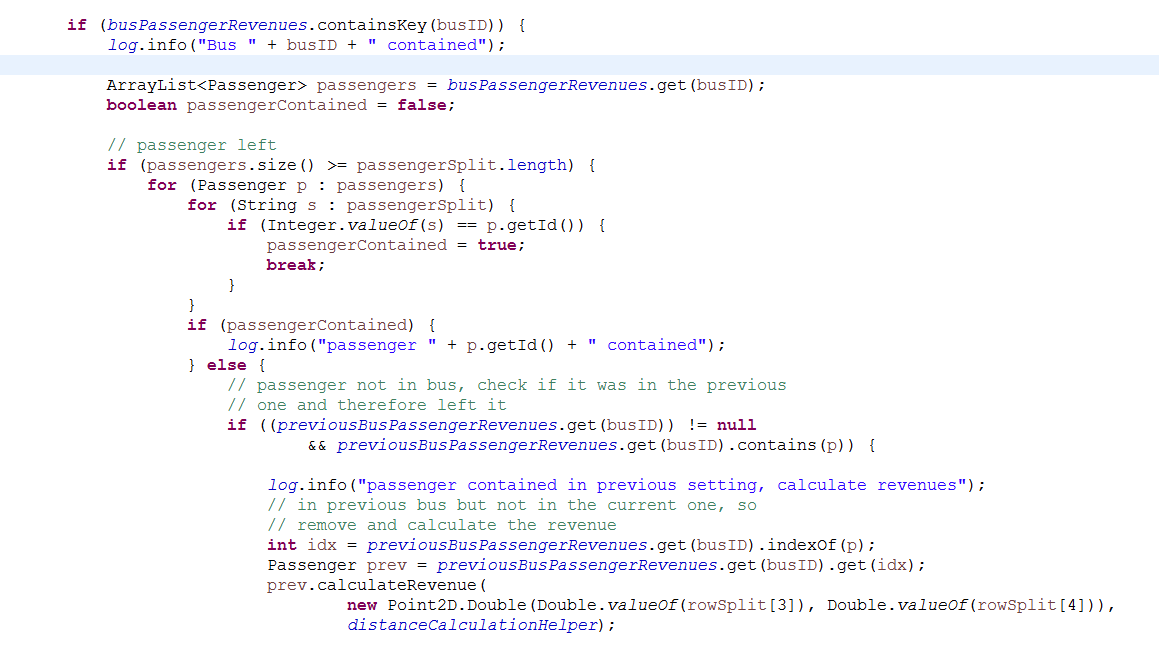
Der liefert Sammlung von JavaRDD’s über die mit der „foreachRDD“ iteriert wird. Jedes RDD ist mehr oder weniger eigener Batch, der wiederum mehrere Zeilen der csv-Datei enthält. Für jede batch wird ein geoJSON Object erstellt, das jede csv-Zeile des batches als eigenes Feature enthält. Ein Feature repräsentiert somit einen bus mit einer gewissen ID und einem gewissen Standpunkt(Lat/Long)

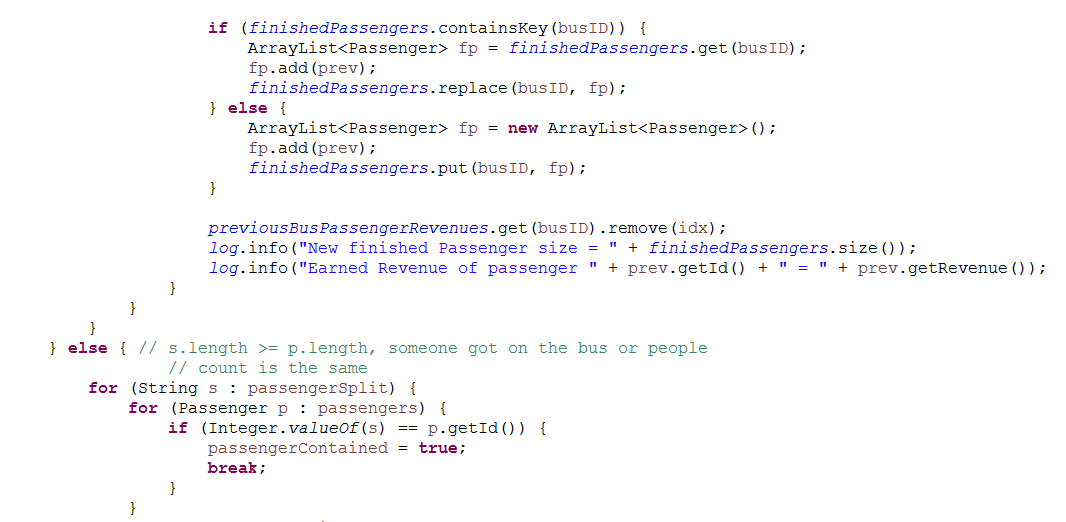


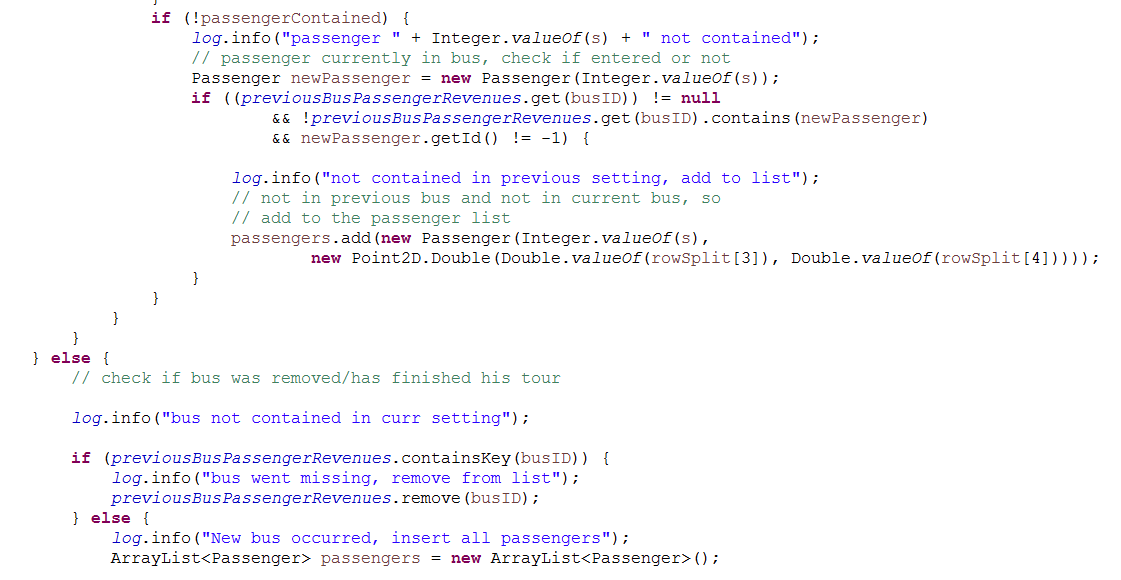
Startet dann einfach das lesen aus dem Stream über den Java StreamingContext

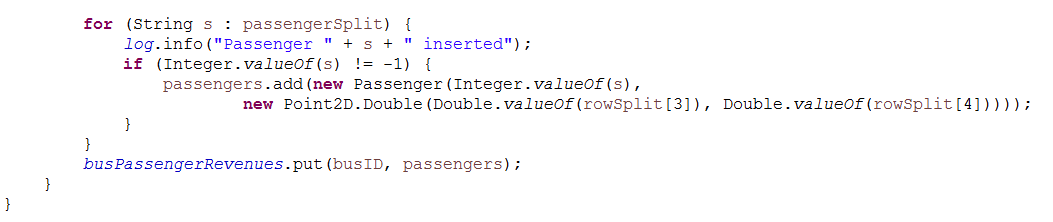
ExtractBusPassengerData: handelt alles ab was mit Passagieren in Bussen zu tun hat. Baut dynamische Datenstruktur (Hashmap) mit BusID und einer Liste von Passagieren auf und passt sie dynamisch je nach aktuellem Stand an. Somit kann dann auf neue Passagiere reagiert werden (werden zur Liste hinzugefügt) und auf Passagiere, die den Bus verlassen (hier wird dann die Revenue Calculation aus der Passenger.java Datei aufgerufen, siehe unten). Zusätzlich wird jeder Passagier, der einen Bus verlassen hat aus der aktuellen Hashmap entfernt und in eine neue Hashmap hinzugefügt, die nur die Passagiere enthält, die einen Bus verlassen haben (gleiche Datenstruktur wie mit aktuellem Bus-Passagier-Zustand)



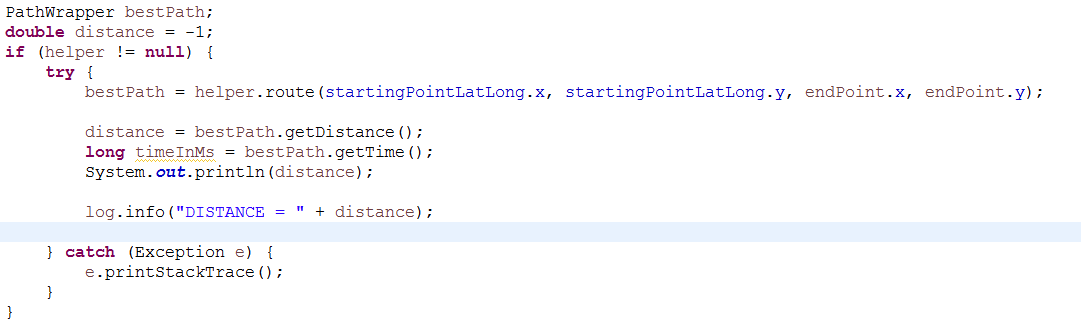








Passenger.java



GraphHopper Helper generiert aus \*.pbf file (das vorher eingelesen wurde) die Route vom Start- zum Endpunkt 🡪 wird in PathWrapper Objekt gespeichert aus dem man sich dann ganz einfach die Distanz über „getDistance“ rausholen kann