Simulatie.java

```
1   package org.tigam.railcab.algoritme;
2
3   import java.io.FileInputStream;
4   import java.io.FileNotFoundException;
5   import java.io.FileOutputStream;
6   import java.io.IOException;
7   import java.io.ObjectInputStream;
8   import java.io.ObjectOutputStream;
9   import java.io.Serializable;
10  import java.util.ArrayList;
11  import java.util.Observable;
12  import java.util.Observer;
13
14  /**
15   * @author Mustapha Bouzaidi
16   *
17   */
18  public class Simulatie implements Serializable, Observer {
19      private final static Simulatie INSTANTIE = new Simulatie();
20      /**
21       *
22       */
23      private static final long serialVersionUID = -7046255051548718419L;
24      private BaanManager baanManager;
25      private ReisManager reisManager;
26      private transient Thread runner;
27      private ArrayList<Reis> voltooideReizen;
28      private long taxiBezettingen, wachtTijden, reisTijden;
29      private int taxiBezettingsAantal, wachtTijdenAantal, reisTijdenAantal;
30
31      private Simulatie() {
32          voltooideReizen = new ArrayList<Reis>();
33          baanManager = new BaanManager();
34          reisManager = new ReisManager(baanManager);
35      }
36
37      /** Geeft de instantie van de Simulatie klasse terug. Gebruikt de singleton pattern.
38       * @return Instantie van simulatie
39       */
40      public static Simulatie getInstantie() {
41          return INSTANTIE;
42      }
43
44      public static void main(String args[]) {
45          Simulatie simulatie = Simulatie.getInstantie();
46          //BaanManager baanManager = simulatie.getBaanManager();
47          //ReisManager reisManager = simulatie.getReisManager();
48          simulatie.getBaanManager().maakStations(5);
49          simulatie.getBaanManager().maakTaxis(2, simulatie);
50          //simulatie.getBaanManager().voegOnbezetteTaxi(new Taxi(simulatie.getBaanManager(), 1, "Taxi 1", simulatie.getBaanManager().getStation("ZuidWTC")), simulatie.getBaanManager().getStation("Zui
51          //simulatie.getBaanManager().voegOnbezetteTaxi(new Taxi(simulatie.getBaanManager(), 2, "Taxi 2", simulatie.getBaanManager().getStation("ZuidWTC")), simulatie.getBaanManager().getStation("Zui
52          //simulatie.getBaanManager().voegOnbezetteTaxi(new Taxi(simulatie.getBaanManager(), 3, "Taxi 3", simulatie.getBaanManager().getStation("ZuidWTC")), simulatie.getBaanManager().getStation("Zui
53          //simulatie.getBaanManager().voegOnbezetteTaxi(new Taxi(simulatie.getBaanManager(), 4, "Taxi 4", simulatie.getBaanManager().getStation("ZuidWTC")), simulatie.getBaanManager().getStation("Zui
54          //simulatie.getBaanManager().voegOnbezetteTaxi(new Taxi(simulatie.getBaanManager(), 5, "Taxi 5", simulatie.getBaanManager().getStation("ZuidWTC")), simulatie.getBaanManager().getStation("Zui
55          try {
56              simulatie.start();
57              for (int i = 0; i < 10; i++) simulatie.getReisManager().addReiziger(simulatie.getBaanManager().getStation("ZuidWTC"), simulatie.getBaanManager().getStation("RAI"));
58              Thread.sleep(5000);
59              //for (int i = 0; i < 10; i++) simulatie.getReisManager().addReiziger(simulatie.getBaanManager().getStation("RAI"), simulatie.getBaanManager().getStation("ZuidWTC"));
60              //for (int i = 0; i < 6; i++) simulatie.getReisManager().addReiziger(simulatie.getBaanManager().getStation("Duivendrecht"), simulatie.getBaanManager().getStation("RAI"));
61              //Thread.sleep(500);
62              //for (int i = 0; i < 8; i++) simulatie.getReisManager().addReiziger(simulatie.getBaanManager().getStation("Duivendrecht"), simulatie.getBaanManager().getStation("ZuidWTC"));
63              //Thread.sleep(1000);
64              // simulatie.versnel();
65              //while (true) {
66              //  Thread.sleep(70000);
67              //  System.out.println(simulatie.gemiddeldeWachtTijd());
68                  simulatie.pauzeer();
69                  Thread.sleep(10000);
```

```
70                  simulatie.hervat();
71              //}
72              //Thread.sleep(5000);
73              //simulatie.stop();
74              //simulatie.opslaan("test.dat");
75              //Thread.sleep(5000);
76              //simulatie.getBaanManager().maakStations(3);
77              //simulatie.getBaanManager().maakTaxis(2, simulatie);
78              //0for (int i = 0; i < 6; i++) simulatie.getReisManager().addReiziger(simulatie.getBaanManager().getStation("RAI"), simulatie.getBaanManager().getStation("ZuidWTC"));
79              //simulatie.start();
80              //simulatie.laden("test.dat");
81              //Thread.sleep(5000);
82              //simulatie.getBaanManager().maakTaxis(2, simulatie);
83          } catch (InterruptedException e) {
84              // TODO Auto-generated catch block
85              e.printStackTrace();
86          }
87      }
88
89      /** Geeft de baanmanager van de simulatie terug.
90       * @return Baanmanager in de simulatie
91       */
92      public synchronized BaanManager getBaanManager() {
93          return baanManager;
94      }
95
96      /** Geeft de reismanager van de simulatie terug.
97       * @return Reismanager in de simulatie
98       */
99      public synchronized ReisManager getReisManager() {
100         return reisManager;
101     }
102
103     /** Voegt de reis aan de lijst met voltooide reizen. Gebruikt vervolgens de reistijd, wachttijden van de reizigers en het aantal reizigers in de reis object om de gemiddelde statistieken aan te
104      * @param reis Voltooide reis
105      */
106     public void voegReis(Reis reis) {
107         voltooideReizen.add(reis);
108
109         taxiBezettingen += (reis.getAantalReizigers() / ReisManager.MAX_REIZIGERS_IN_TAXI) * 100;
110         taxiBezettingsAantal++;
111
112         for (Reiziger r : reis.getReizigers()) {
113             wachtTijden += r.getWachtTijd();
114             wachtTijdenAantal++;
115         }
116
117         reisTijden += reis.getReisTijd();
118         reisTijdenAantal++;
119     }
120
121     /** Geeft de gemiddelde taxi bezetting (gemiddeld aantal reizigers in taxi) terug.
122      * @return Taxibezetting
123      */
124     public synchronized long gemiddeldeTaxiBezetting() {
125         if (taxiBezettingsAantal > 0) return taxiBezettingen / taxiBezettingsAantal;
126         else return 0;
127     }
128
129     /** Geeft de gemiddelde wachttijd van reizigers terug.
130      * @return Gemiddelde wachttijd van reizigers.
131      */
132     public synchronized long gemiddeldeWachtTijd() {
133         if (wachtTijdenAantal > 0) return wachtTijden / wachtTijdenAantal;
134         else return 0;
135     }
136
137     /** Geeft de gemiddelde reistijd van reizen.
138      * @return Gemiddelde reistijd van voltooide reizen
139      */
140     public synchronized long gemiddeldeReisTijd() {
```

```
141         if (reisTijdenAantal > 0) return reisTijden / reisTijdenAantal;
142         else return 0;
143     }
144 
145     /**
146      * Start de simulatie door de reismanager aan een nieuwe thread te koppelen en te starten.
147      */
148     public synchronized void start() {
149         runner = new Thread(reisManager);
150         reisManager.setStatus(ReisManager.START);
151         runner.start();
152     }
153 
154     /**
155      * Stopt de simulatie. Status van de reismanager wordt op stop gezet en er wordt gewacht tot de thread is uitgestorven.
156      */
157     public synchronized void stop() {
158         reisManager.setStatus(ReisManager.STOP);
159         while (runner.isAlive()) {
160             try {
161                 Thread.sleep(1);
162             } catch (InterruptedException e) {
163                 e.printStackTrace();
164             }
165         }
166         baanManager = new BaanManager();
167         reisManager = new ReisManager(baanManager);
168 
169         System.out.println("Simulatie gestopt!");
170     }
171 
172     /**
173      * Pauzeert de simulatie.
174      */
175     public void pauzeer() {
176         synchronized (runner) { reisManager.pauzeer(); }
177         System.out.println("Gepauzeerd");
178     }
179 
180     /**
181      * Hervat de simulatie.
182      */
183     public void hervat() {
184         synchronized (reisManager) {
185         reisManager.hervat();
186         reisManager.notify();
187         }
188         System.out.println("Hervat");
189     }
190 
191     /**
192      * Simulatie wordt versneld of genormaliseerd.
193      */
194     public synchronized void versnel() {
195         if (ReisManager.getVersnelFactor() <= 1) ReisManager.setVersnelFactor(5);
196         else ReisManager.setVersnelFactor(1);
197     }
198 
199     /** Laad een eerder opgeslagen simulatie d.m.v. serialisatie.
200      * De huidige simulatie moet hierbij wel gepauzeerd zijn.
201      * Gebruikt een ObjectInptutStream in combinatie met een FileInputStream.
202      * @param bestand Bestandsnaam (en eventueel een pad) van een opgeslagen bestand
203      */
204     public synchronized void laden(String bestand) {
205         if (reisManager.getStatus() == ReisManager.PAUZE) {
206             ObjectInputStream objectInputStream;
207             try {
208                 objectInputStream = new ObjectInputStream(new FileInputStream(bestand));
209                 reisManager = null;
210                 baanManager = null;
211                 reisManager = (ReisManager) objectInputStream.readObject();
```

```
212                 baanManager = reisManager.getBaanManager();
213                 objectInputStream.close();
214             } catch (IOException e) {
215                 // TODO Auto-generated catch block
216                 e.printStackTrace();
217             } catch (ClassNotFoundException e) {
218                 // TODO Auto-generated catch block
219                 e.printStackTrace();
220             }
221             runner = new Thread(reisManager);
222             runner.start();
223         }
224     }
225
226     /** Slaat een lopende simulatie op naar de opgegeven bestand d.m.v. serialisatie.
227      * De simulatie wordt tijdelijk gepauzeerd om de simulatie naar het bestand te schrijven.
228      * Gebruikt een ObjectOutputStream in combinatie met een FileOutPutStream.
229      * @param bestand Bestandsnaam (en eventueel een pad) waarnaar de simulatie weggeschreven moet worden.
230      */
231     public synchronized void opslaan(String bestand) {
232         ObjectOutputStream objectOutputStream;
233         try {
234             objectOutputStream = new ObjectOutputStream(new FileOutputStream(bestand));
235             pauzeer();
236             objectOutputStream.writeObject(reisManager);
237             objectOutputStream.close();
238         } catch (FileNotFoundException e) {
239             // TODO Auto-generated catch block
240             e.printStackTrace();
241         } catch (IOException e) {
242             // TODO Auto-generated catch block
243             e.printStackTrace();
244         }
245         hervat();
246     }
247
248     public void update(Observable arg0, Object arg1) {
249         Taxi t = (Taxi) arg0;
250         System.out.println("Observer geupdate, " + t.getNaam());
251     }
252 }
253
```

BaanManager.java

```
1     package org.tigam.railcab.algoritme;
2     import java.io.Serializable;
3     import java.util.ArrayList;
4     import java.util.HashMap;
5     import java.util.NoSuchElementException;
6     import java.util.Observer;
7
8     /** De <code>BaanManager</code> bevat de baanstructuur, stations en taxi's. De baanstructuur bevat hulppmiddelen voor de baan zoals afstandsberekening, controleren op aanrijdingen of het schakelen e
9      * @author Mustapha Bouzaidi
10     *
11     */
12    public class BaanManager implements Serializable {
13        private static final long serialVersionUID = -548428743269703671 4L;
14        /**
15         * De minimale afstand waarna een taxi een wissel mag schakelen.
16         */
17        public static final int MIN_AFSTAND_TOT_WISSEL = 500;
18        /**
19         * De minimale afstand dat taxi's van elkaar moeten nemen.
20         */
21        public static final int MIN_AFSTAND_TOT_TAXI = 500;
22
23        private int wachtTijdStation = 10000;
24        private int taxiSnelheid = 22;
25        private int taxiTeller;
26        private Spoor sporen;
27        private ArrayList<Station> stations;
28        private ArrayList<Taxi> taxis;
29        private HashMap<Taxi, Wisselspoor> wisselLijst;
30
31        /**
32         * Creëert een instantie van de baanmanager zonder een baanstructuur, stations en taxi's.
33         */
34        public BaanManager() {
35            stations = new ArrayList<Station>();
36            taxis = new ArrayList<Taxi>();
37            wisselLijst = new HashMap<Taxi, Wisselspoor>();
38            taxiTeller = 0;
39        }
40
41        /** Geeft de baanstructuur terug.
42         * @return De baanstructuur, beginnend bij de eerste spoordeel
43         */
44        public Spoor getSporen() {
45            return sporen;
46        }
47
48        /** Plaatst de baanstructuur die door de baanmanager gebruikt wordt.
49         * @param spoor De baanstructuur
50         */
51        public void setSporen(Spoor spoor) {
52            this.sporen = spoor;
53        }
54
55        /** Geeft alle taxi's in de baanmanager terug.
56         * @return Alle taxi's in de baanmanager
57         */
58        public ArrayList<Taxi> getTaxis() {
59            return taxis;
60        }
61
62        /** Plaatst de taxi's die door de baanmanager gebruikt worden.
63         * @param taxis Lijst met taxi objecten
64         */
65        public void setTaxi(ArrayList<Taxi> taxis) {
66            this.taxis = taxis;
67        }
68
69        /** Voegt een taxi object toe die in de baanmanager gebruikt wordt.
```

```
70          * @param taxi De taxi
71          */
72         public void voegTaxi(Taxi taxi) {
73             taxis.add(taxi);
74         }
75
76         /** Voegt een onbezette taxi in de baanmanager en aan de betreffende station.
77          * @param taxi De onbezette taxi
78          * @param station Het station waar de onbezette taxi moet staan
79          */
80         public synchronized void voegOnbezetteTaxi(Taxi taxi, Station station) {
81             taxi.setStatus(TaxiStatus.ONBEZET);
82             voegTaxi(taxi);
83             station.voegTaxiIn(taxi);
84         }
85
86         /** plaatst de stations in de baanmanager
87          * @param stations De stations
88          */
89         public void setStations(ArrayList<Station> stations) {
90             this.stations = stations;
91         }
92
93         /** Geeft de namen van alle stations in de baanmanager terug.
94          * @return namen van alle stations in
95          */
96         public String[] getStations() {
97             String[] stationsNamen = new String[stations.size()];
98             for (int i = 0; i < stations.size(); i++)
99                 stationsNamen[i] = stations.get(i).getNaam();
100            return stationsNamen;
101        }
102
103        /** Geeft alle beschikbare taxi's terug.
104         * @return alle beschikbare (onbezette) taxi's
105         */
106        public ArrayList<Taxi> getBeschikbareTaxis() {
107            ArrayList<Taxi> oTaxis = new ArrayList<Taxi>();
108            for (Taxi t : taxis) {
109                if (t.getStatus() == TaxiStatus.ONBEZET) {
110                    try {
111                        if (t.getReis() == null) { }
112                    } catch (RuntimeException e) {
113                        oTaxis.add(t);
114                    }
115                }
116            }
117            return oTaxis;
118        }
119
120        /** Bepaalt de afstand tussen 2 stations.
121         * @param begin Station waar de afstand vanaf bepaald wordt
122         * @param eind Station waar de afstand tot bepaald wordt
123         * @return Afstand tussen het begin en eindstation
124         */
125        public long bepaalAfstand(Station begin, Station eind) {
126            long afstand = 0;
127            if (begin == eind) return afstand;
128            boolean overslaan = false, beginBereikt = false;
129            for (Spoor spoor : sporen) {
130                if (!overslaan) {
131                    if (spoor.getType() == SpoorType.WISSEL_SPOOR && ((Wisselspoor)spoor).getRichting() == Wisselspoor.RICHTING_VOOR_ZIJSPOOR) {
132                        Wisselspoor wSpoor = (Wisselspoor) spoor;
133                        if (getStation(wSpoor) == begin) {
134                            Stationspoor sSpoor = (Stationspoor) wSpoor.getZijSpoor();
135                            afstand += sSpoor.getLengte() - sSpoor.getStation().getPositie();
136                            afstand += sSpoor.volgende().getLengte();
137                            overslaan = true;
138                            beginBereikt = true;
139                        }
140                        else if (getStation(wSpoor) == eind && beginBereikt) {
```

```
141                         afstand += wSpoor.getLengte();
142                         afstand += wSpoor.getZijSpoor().getLengte() - ((Stationspoor)wSpoor.getZijSpoor()).getStation().getPositie();
143                         return afstand;
144                     }
145                 }
146                 else if (beginBereikt) {
147                     afstand += spoor.getLengte();
148                 }
149             }
150             else overslaan = false;
151         }
152         return afstand;
153     }
154 
155     /** Geeft het station terug met de betreffende naam.
156      * @param naam De naam van de station
157      * @return De station met de opgegeven naam
158      */
159     public Station getStation(String naam) {
160         for (Station s : stations) {
161             if (s.getNaam().equals(naam))
162                 return s;
163         }
164         return null;
165     }
166 
167     /** Maakt het aantal stations aan en de baanstructuur eromheen.
168      * Kan minimaal 2 en maximaal 8 stations aanmaken. Hoofdbaan is altijd 24 km breed.
169      * Zijbanen zijn altijd 1 km breed incl. wisselsporen. De wisselsporen zijn elk 50 meter breed.
170      * @param aantal De hoeveelheid stations die aangemaakt moeten worden.
171      * @return Is waar als de stations en baanstructuur is aangemaakt en anders onwaar.
172      */
173     public synchronized boolean maakStations(int aantal) {
174         Wisselspoor ws[] = new Wisselspoor[aantal * 2];
175         for (int i = 0; i < aantal; i++) {
176             ws[i + i] = new Wisselspoor("WS" + (i + i + 1), 50, Wisselspoor.RICHTING_VOOR_ZIJSPOOR);
177             ws[i + i + 1] = new Wisselspoor("WS" + (i + i + 2), 50, Wisselspoor.RICHTING_NA_ZIJSPOOR);
178         }
179 
180         Spoor spoor1, spoor2, spoor3, spoor4, spoor5, spoor6, spoor7, spoor8,
181         spoor9, spoor10, spoor11, spoor12, spoor13, spoor14, spoor15, spoor16, spoor17;
182         Stationspoor sSpoor1, sSpoor2, sSpoor3, sSpoor4, sSpoor5, sSpoor6, sSpoor7, sSpoor8;
183 
184         Station station1, station2, station3, station4, station5, station6, station7, station8;
185 
186         switch (aantal) {
187         case 2:
188             station1 = new Station(1, "RAI", 450);
189             station2 = new Station(2, "Duivendrecht", 450);
190             stations.add(station1);
191             stations.add(station2);
192 
193             spoor1 = new Spoor("HS1", 500);
194             spoor1.setNaSpoor(ws[0]);
195 
196             spoor2 = new Spoor("HS2", 900, ws[0], ws[1]);
197             sSpoor1 = new Stationspoor("SS1", 900, ws[0], ws[1], station1);
198             spoor3 = new Spoor("HS3", 11000, ws[1], ws[2]);
199             spoor4 = new Spoor("HS4", 900, ws[2], ws[3]);
200             sSpoor2 = new Stationspoor("SS2", 900, ws[2], ws[3], station2);
201             spoor5 = new Spoor("HS5", 10500, ws[3], spoor1);
202             ws[0].setZijSpoor(sSpoor1);
203             ws[0].setNaSpoor(spoor2);
204             ws[1].setZijSpoor(sSpoor1);
205             ws[1].setNaSpoor(spoor3);
206             ws[2].setZijSpoor(sSpoor2);
207             ws[2].setNaSpoor(spoor4);
208             ws[3].setZijSpoor(sSpoor2);
209             ws[3].setNaSpoor(spoor5);
210             setSporen(spoor1);
211             System.out.println("Baanstructuur aangemaakt!");
```

```
212             return true;
213         case 3:
214             station1 = new Station(1, "RAI", 450);
215             station2 = new Station(2, "Duivendrecht", 450);
216             station3 = new Station(3, "ZuidWTC", 450);
217             stations.add(station1);
218             stations.add(station2);
219             stations.add(station3);
220 
221             spoor1 = new Spoor("HS1", 3500);
222             spoor1.setNaSpoor(ws[0]);
223 
224             spoor2 = new Spoor("HS2", 900, ws[0], ws[1]);
225             sSpoor1 = new Stationspoor("SS1", 900, ws[0], ws[1], station1);
226             spoor3 = new Spoor("HS3", 8000, ws[1], ws[2]);
227             spoor4 = new Spoor("HS4", 900, ws[2], ws[3]);
228             sSpoor2 = new Stationspoor("SS2", 900, ws[2], ws[3], station2);
229             spoor5 = new Spoor("HS5", 5000, ws[3], ws[4]);
230             spoor6 = new Spoor("HS6", 900, ws[4], ws[5]);
231             sSpoor3 = new Stationspoor("SS3", 900, ws[4], ws[5], station3);
232             spoor7 = new Spoor("HS7", 4500, ws[5], spoor1);
233 
234             ws[0].setZijSpoor(sSpoor1);
235             ws[0].setNaSpoor(spoor2);
236             ws[1].setZijSpoor(sSpoor1);
237             ws[1].setNaSpoor(spoor3);
238             ws[2].setZijSpoor(sSpoor2);
239             ws[2].setNaSpoor(spoor4);
240             ws[3].setZijSpoor(sSpoor2);
241             ws[3].setNaSpoor(spoor5);
242             ws[4].setZijSpoor(sSpoor3);
243             ws[4].setNaSpoor(spoor6);
244             ws[5].setZijSpoor(sSpoor3);
245             ws[5].setNaSpoor(spoor7);
246 
247             setSporen(spoor1);
248             System.out.println("Baanstructuur aangemaakt!");
249             return true;
250         case 4:
251             station1 = new Station(1, "Muiderpoort", 450);
252             station2 = new Station(2, "ZuidWTC", 450);
253             station3 = new Station(3, "RAI", 450);
254             station4 = new Station(4, "Duivendrecht", 450);
255             stations.add(station1);
256             stations.add(station2);
257             stations.add(station3);
258             stations.add(station4);
259 
260             spoor1 = new Spoor("HS1", 500);
261             spoor1.setNaSpoor(ws[0]);
262 
263             spoor2 = new Spoor("HS2", 900, ws[0], ws[1]);
264             sSpoor1 = new Stationspoor("SS1", 900, ws[0], ws[1], station1);
265             spoor3 = new Spoor("HS3", 5000, ws[1], ws[2]);
266             spoor4 = new Spoor("HS4", 900, ws[2], ws[3]);
267             sSpoor2 = new Stationspoor("SS2", 900, ws[2], ws[3], station2);
268             spoor5 = new Spoor("HS5", 5000, ws[3], ws[4]);
269             spoor6 = new Spoor("HS6", 900, ws[4], ws[5]);
270             sSpoor3 = new Stationspoor("SS3", 900, ws[4], ws[5], station3);
271             spoor7 = new Spoor("HS7", 5000, ws[5], ws[6]);
272             spoor8 = new Spoor("HS8", 900, ws[6], ws[7]);
273             sSpoor4 = new Stationspoor("SS4", 900, ws[6], ws[7], station4);
274             spoor9 = new Spoor("HS9", 4500, ws[7], spoor1);
275 
276             ws[0].setZijSpoor(sSpoor1);
277             ws[0].setNaSpoor(spoor2);
278             ws[1].setZijSpoor(sSpoor1);
279             ws[1].setNaSpoor(spoor3);
280             ws[2].setZijSpoor(sSpoor2);
281             ws[2].setNaSpoor(spoor4);
282             ws[3].setZijSpoor(sSpoor2);
```

```
283             ws[3].setNaSpoor(spoor5);
284             ws[4].setZijSpoor(sSpoor3);
285             ws[4].setNaSpoor(spoor6);
286             ws[5].setZijSpoor(sSpoor3);
287             ws[5].setNaSpoor(spoor7);
288             ws[6].setZijSpoor(sSpoor4);
289             ws[6].setNaSpoor(spoor8);
290             ws[7].setZijSpoor(sSpoor4);
291             ws[7].setNaSpoor(spoor9);
292
293             setSporen(spoor1);
294             System.out.println("Baanstructuur aangemaakt!");
295             return true;
296         case 5:
297             station1 = new Station(1, "AmsterdamCS", 450);
298             station2 = new Station(2, "Muiderpoort", 450);
299             station3 = new Station(3, "ZuidWTC", 450);
300             station4 = new Station(4, "RAI", 450);
301             station5 = new Station(5, "Duivendrecht", 450);
302             stations.add(station1);
303             stations.add(station2);
304             stations.add(station3);
305             stations.add(station4);
306             stations.add(station5);
307
308             spoor1 = new Spoor("HS1", 3500);
309             spoor1.setNaSpoor(ws[0]);
310
311             spoor2 = new Spoor("HS2", 900, ws[0], ws[1]);
312             sSpoor1 = new Stationspoor("SS1", 900, ws[0], ws[1], station1);
313             spoor3 = new Spoor("HS3", 5000, ws[1], ws[2]);
314             spoor4 = new Spoor("HS4", 900, ws[2], ws[3]);
315             sSpoor2 = new Stationspoor("SS2", 900, ws[2], ws[3], station2);
316             spoor5 = new Spoor("HS5", 2000, ws[3], ws[4]);
317             spoor6 = new Spoor("HS6", 900, ws[4], ws[5]);
318             sSpoor3 = new Stationspoor("SS3", 900, ws[4], ws[5], station3);
319             spoor7 = new Spoor("HS7", 5000, ws[5], ws[6]);
320             spoor8 = new Spoor("HS8", 900, ws[6], ws[7]);
321             sSpoor4 = new Stationspoor("SS4", 900, ws[6], ws[7], station4);
322             spoor9 = new Spoor("HS9", 2000, ws[7], ws[8]);
323             spoor10 = new Spoor("HS10", 900, ws[8], ws[9]);
324             sSpoor5 = new Stationspoor("SS5", 900, ws[8], ws[9], station5);
325             spoor11 = new Spoor("HS11", 1500, ws[9], spoor1);
326
327             ws[0].setZijSpoor(sSpoor1);
328             ws[0].setNaSpoor(spoor2);
329             ws[1].setZijSpoor(sSpoor1);
330             ws[1].setNaSpoor(spoor3);
331             ws[2].setZijSpoor(sSpoor2);
332             ws[2].setNaSpoor(spoor4);
333             ws[3].setZijSpoor(sSpoor2);
334             ws[3].setNaSpoor(spoor5);
335             ws[4].setZijSpoor(sSpoor3);
336             ws[4].setNaSpoor(spoor6);
337             ws[5].setZijSpoor(sSpoor3);
338             ws[5].setNaSpoor(spoor7);
339             ws[6].setZijSpoor(sSpoor4);
340             ws[6].setNaSpoor(spoor8);
341             ws[7].setZijSpoor(sSpoor4);
342             ws[7].setNaSpoor(spoor9);
343             ws[8].setZijSpoor(sSpoor5);
344             ws[8].setNaSpoor(spoor10);
345             ws[9].setZijSpoor(sSpoor5);
346             ws[9].setNaSpoor(spoor11);
347
348             setSporen(spoor1);
349             System.out.println("Baanstructuur aangemaakt!");
350             return true;
351         case 6:
352             station1 = new Station(1, "Sloterdijk", 450);
353             station2 = new Station(2, "AmsterdamCS", 450);
```

```
354             station3 = new Station(3, "Muiderpoort", 450);
355             station4 = new Station(4, "ZuidWTC", 450);
356             station5 = new Station(5, "RAI", 450);
357             station6 = new Station(6, "Duivendrecht", 450);
358             stations.add(station1);
359             stations.add(station2);
360             stations.add(station3);
361             stations.add(station4);
362             stations.add(station5);
363             stations.add(station6);
364
365             spoor1 = new Spoor("HS1", 500);
366             spoor1.setNaSpoor(ws[0]);
367
368             spoor2 = new Spoor("HS2", 900, ws[0], ws[1]);
369             sSpoor1 = new Stationspoor("SS1", 900, ws[0], ws[1], station1);
370             spoor3 = new Spoor("HS3", 5000, ws[1], ws[2]);
371             spoor4 = new Spoor("HS4", 900, ws[2], ws[3]);
372             sSpoor2 = new Stationspoor("SS2", 900, ws[2], ws[3], station2);
373             spoor5 = new Spoor("HS5", 2000, ws[3], ws[4]);
374             spoor6 = new Spoor("HS6", 900, ws[4], ws[5]);
375             sSpoor3 = new Stationspoor("SS3", 900, ws[4], ws[5], station3);
376             spoor7 = new Spoor("HS7", 2000, ws[5], ws[6]);
377             spoor8 = new Spoor("HS8", 900, ws[6], ws[7]);
378             sSpoor4 = new Stationspoor("SS4", 900, ws[6], ws[7], station4);
379             spoor9 = new Spoor("HS9", 5000, ws[7], ws[8]);
380             spoor10 = new Spoor("HS10", 900, ws[8], ws[9]);
381             sSpoor5 = new Stationspoor("SS5", 900, ws[8], ws[9], station5);
382             spoor11 = new Spoor("HS11", 2000, ws[9], ws[10]);
383             spoor12 = new Spoor("HS12", 900, ws[10], ws[11]);
384             sSpoor6 = new Stationspoor("SS6", 900, ws[10], ws[11], station6);
385             spoor13 = new Spoor("HS13", 1500, ws[11], spoor1);
386
387             ws[0].setZijSpoor(sSpoor1);
388             ws[0].setNaSpoor(spoor2);
389             ws[1].setZijSpoor(sSpoor1);
390             ws[1].setNaSpoor(spoor3);
391             ws[2].setZijSpoor(sSpoor2);
392             ws[2].setNaSpoor(spoor4);
393             ws[3].setZijSpoor(sSpoor2);
394             ws[3].setNaSpoor(spoor5);
395             ws[4].setZijSpoor(sSpoor3);
396             ws[4].setNaSpoor(spoor6);
397             ws[5].setZijSpoor(sSpoor3);
398             ws[5].setNaSpoor(spoor7);
399             ws[6].setZijSpoor(sSpoor4);
400             ws[6].setNaSpoor(spoor8);
401             ws[7].setZijSpoor(sSpoor4);
402             ws[7].setNaSpoor(spoor9);
403             ws[8].setZijSpoor(sSpoor5);
404             ws[8].setNaSpoor(spoor10);
405             ws[9].setZijSpoor(sSpoor5);
406             ws[9].setNaSpoor(spoor11);
407             ws[10].setZijSpoor(sSpoor6);
408             ws[10].setNaSpoor(spoor12);
409             ws[11].setZijSpoor(sSpoor6);
410             ws[11].setNaSpoor(spoor13);
411
412             setSporen(spoor1);
413             System.out.println("Baanstructuur aangemaakt!");
414             return true;
415         case 7:
416             station1 = new Station(1, "Lelylaan", 450);
417             station2 = new Station(2, "Sloterdijk", 450);
418             station3 = new Station(3, "AmsterdamCS", 450);
419             station4 = new Station(4, "Muiderpoort", 450);
420             station5 = new Station(5, "ZuidWTC", 450);
421             station6 = new Station(6, "RAI", 450);
422             station7 = new Station(7, "Duivendrecht", 450);
423             stations.add(station1);
424             stations.add(station2);
```

```
425             stations.add(station3);
426             stations.add(station4);
427             stations.add(station5);
428             stations.add(station6);
429             stations.add(station7);
430
431             spoor1 = new Spoor("HS1", 500);
432             spoor1.setNaSpoor(ws[0]);
433
434             spoor2 = new Spoor("HS2", 900, ws[0], ws[1]);
435             sSpoor1 = new Stationspoor("SS1", 900, ws[0], ws[1], station1);
436             spoor3 = new Spoor("HS3", 2000, ws[1], ws[2]);
437             spoor4 = new Spoor("HS4", 900, ws[2], ws[3]);
438             sSpoor2 = new Stationspoor("SS2", 900, ws[2], ws[3], station2);
439             spoor5 = new Spoor("HS5", 2000, ws[3], ws[4]);
440             spoor6 = new Spoor("HS6", 900, ws[4], ws[5]);
441             sSpoor3 = new Stationspoor("SS3", 900, ws[4], ws[5], station3);
442             spoor7 = new Spoor("HS7", 2000, ws[5], ws[6]);
443             spoor8 = new Spoor("HS8", 900, ws[6], ws[7]);
444             sSpoor4 = new Stationspoor("SS4", 900, ws[6], ws[7], station4);
445             spoor9 = new Spoor("HS9", 2000, ws[7], ws[8]);
446             spoor10 = new Spoor("HS10", 900, ws[8], ws[9]);
447             sSpoor5 = new Stationspoor("SS5", 900, ws[8], ws[9], station5);
448             spoor11 = new Spoor("HS11", 5000, ws[9], ws[10]);
449             spoor12 = new Spoor("HS12", 900, ws[10], ws[11]);
450             sSpoor6 = new Stationspoor("SS6", 900, ws[10], ws[11], station6);
451             spoor13 = new Spoor("HS13", 2000, ws[11], ws[12]);
452             spoor14 = new Spoor("HS14", 900, ws[12], ws[13]);
453             sSpoor7 = new Stationspoor("SS7", 900, ws[12], ws[13], station7);
454             spoor15 = new Spoor("HS15", 1500, ws[13], spoor1);
455
456             ws[0].setZijSpoor(sSpoor1);
457             ws[0].setNaSpoor(spoor2);
458             ws[1].setZijSpoor(sSpoor1);
459             ws[1].setNaSpoor(spoor3);
460             ws[2].setZijSpoor(sSpoor2);
461             ws[2].setNaSpoor(spoor4);
462             ws[3].setZijSpoor(sSpoor2);
463             ws[3].setNaSpoor(spoor5);
464             ws[4].setZijSpoor(sSpoor3);
465             ws[4].setNaSpoor(spoor6);
466             ws[5].setZijSpoor(sSpoor3);
467             ws[5].setNaSpoor(spoor7);
468             ws[6].setZijSpoor(sSpoor4);
469             ws[6].setNaSpoor(spoor8);
470             ws[7].setZijSpoor(sSpoor4);
471             ws[7].setNaSpoor(spoor9);
472             ws[8].setZijSpoor(sSpoor5);
473             ws[8].setNaSpoor(spoor10);
474             ws[9].setZijSpoor(sSpoor5);
475             ws[9].setNaSpoor(spoor11);
476             ws[10].setZijSpoor(sSpoor6);
477             ws[10].setNaSpoor(spoor12);
478             ws[11].setZijSpoor(sSpoor6);
479             ws[11].setNaSpoor(spoor13);
480             ws[12].setZijSpoor(sSpoor7);
481             ws[12].setNaSpoor(spoor14);
482             ws[13].setZijSpoor(sSpoor7);
483             ws[13].setNaSpoor(spoor15);
484
485             setSporen(spoor1);
486             System.out.println("Baanstructuur aangemaakt!");
487             return true;
488         case 8:
489             station1 = new Station(1, "Amstel", 450);
490             station2 = new Station(2, "Lelylaan", 450);
491             station3 = new Station(3, "Sloterdijk", 450);
492             station4 = new Station(4, "AmsterdamCS", 450);
493             station5 = new Station(5, "Muiderpoort", 450);
494             station6 = new Station(6, "ZuidWTC", 450);
495             station7 = new Station(7, "RAI", 450);
```

```
496             station8 = new Station(8, "Duivendrecht", 450);
497             stations.add(station1);
498             stations.add(station2);
499             stations.add(station3);
500             stations.add(station4);
501             stations.add(station5);
502             stations.add(station6);
503             stations.add(station7);
504             stations.add(station8);
505
506             spoor1 = new Spoor("HS1", 500);
507             spoor1.setNaSpoor(ws[0]);
508
509             spoor2 = new Spoor("HS2", 900, ws[0], ws[1]);
510             sSpoor1 = new Stationspoor("SS1", 900, ws[0], ws[1], station1);
511             spoor3 = new Spoor("HS3", 2000, ws[1], ws[2]);
512             spoor4 = new Spoor("HS4", 900, ws[2], ws[3]);
513             sSpoor2 = new Stationspoor("SS2", 900, ws[2], ws[3], station2);
514             spoor5 = new Spoor("HS5", 2000, ws[3], ws[4]);
515             spoor6 = new Spoor("HS6", 900, ws[4], ws[5]);
516             sSpoor3 = new Stationspoor("SS3", 900, ws[4], ws[5], station3);
517             spoor7 = new Spoor("HS7", 2000, ws[5], ws[6]);
518             spoor8 = new Spoor("HS8", 900, ws[6], ws[7]);
519             sSpoor4 = new Stationspoor("SS4", 900, ws[6], ws[7], station4);
520             spoor9 = new Spoor("HS9", 2000, ws[7], ws[8]);
521             spoor10 = new Spoor("HS10", 900, ws[8], ws[9]);
522             sSpoor5 = new Stationspoor("SS5", 900, ws[8], ws[9], station5);
523             spoor11 = new Spoor("HS11", 2000, ws[9], ws[10]);
524             spoor12 = new Spoor("HS12", 900, ws[10], ws[11]);
525             sSpoor6 = new Stationspoor("SS6", 900, ws[10], ws[11], station6);
526             spoor13 = new Spoor("HS13", 2000, ws[11], ws[12]);
527             spoor14 = new Spoor("HS14", 900, ws[12], ws[13]);
528             sSpoor7 = new Stationspoor("SS7", 900, ws[12], ws[13], station7);
529             spoor15 = new Spoor("HS15", 2000, ws[13], ws[14]);
530             spoor16 = new Spoor("HS16", 900, ws[14], ws[15]);
531             sSpoor8 = new Stationspoor("SS8", 900, ws[14], ws[15], station8);
532             spoor17 = new Spoor("HS17", 1500, ws[15], spoor1);
533
534             ws[0].setZijSpoor(sSpoor1);
535             ws[0].setNaSpoor(spoor2);
536             ws[1].setZijSpoor(sSpoor1);
537             ws[1].setNaSpoor(spoor3);
538             ws[2].setZijSpoor(sSpoor2);
539             ws[2].setNaSpoor(spoor4);
540             ws[3].setZijSpoor(sSpoor2);
541             ws[3].setNaSpoor(spoor5);
542             ws[4].setZijSpoor(sSpoor3);
543             ws[4].setNaSpoor(spoor6);
544             ws[5].setZijSpoor(sSpoor3);
545             ws[5].setNaSpoor(spoor7);
546             ws[6].setZijSpoor(sSpoor4);
547             ws[6].setNaSpoor(spoor8);
548             ws[7].setZijSpoor(sSpoor4);
549             ws[7].setNaSpoor(spoor9);
550             ws[8].setZijSpoor(sSpoor5);
551             ws[8].setNaSpoor(spoor10);
552             ws[9].setZijSpoor(sSpoor5);
553             ws[9].setNaSpoor(spoor11);
554             ws[10].setZijSpoor(sSpoor6);
555             ws[10].setNaSpoor(spoor12);
556             ws[11].setZijSpoor(sSpoor6);
557             ws[11].setNaSpoor(spoor13);
558             ws[12].setZijSpoor(sSpoor7);
559             ws[12].setNaSpoor(spoor14);
560             ws[13].setZijSpoor(sSpoor7);
561             ws[13].setNaSpoor(spoor15);
562             ws[14].setZijSpoor(sSpoor8);
563             ws[14].setNaSpoor(spoor16);
564             ws[15].setZijSpoor(sSpoor8);
565             ws[15].setNaSpoor(spoor17);
566
```

```
567             setSporen(spoor1);
568             System.out.println("Baanstructuur aangemaakt!");
569             return true;
570         }
571         return false;
572     }
573
574     /** Maakt onbezette taxi's aan die verdeelt worden over de beschikbare stations.
575      * De observer wordt toegevoegd aan de taxi's, maar een null waarde kan als argument bij de observer parameter worden meegegeven wanneer dit niet gewenst is.
576      * @param aantal Aantal taxi's om aan te maken.
577      * @param observer De observer dat de aan de taxi's wordt toegevoegd
578      * @throws RuntimeException Wanneer er geen stations beschikbaar zijn
579      */
580     public synchronized void maakTaxis(int aantal, Observer observer) throws RuntimeException {
581         if (stations.isEmpty()) throw new RuntimeException("Geen stations beschikbaar om taxi's te plaatsen");
582         for (int i = 0; i < aantal; i++) {
583             Station s = stations.get(i % stations.size());
584             taxiTeller++;
585             Taxi t;
586             if (observer == null) t = new Taxi(this, taxiTeller, "Taxi " + taxiTeller, s);
587             else t = new Taxi(this, taxiTeller, "Taxi " + taxiTeller, s, observer);
588             voegOnbezetteTaxi(t, s);
589         }
590     }
591
592     /** Geeft het station met de opgegeven ID terug.
593      * @param ID De ID nummer van het station
594      * @return Station met de opgegeven ID
595      * @throws RuntimeException Wanneer het station met de opgegeven ID niet gevonden kan worden
596      */
597     public Station getStation(int ID) throws RuntimeException {
598         for (Station s : stations) {
599             if (s.getID() == ID)
600                 return s;
601         }
602         throw new RuntimeException("Geen station gevonden met ID: " + ID);
603     }
604
605     /** Geeft het aantal taxi's terug.
606      * @return aantal taxi's
607      */
608     public int getAantalTaxis() {
609         return taxis.size();
610     }
611
612     /** Geeft het aantal stations terug.
613      * @return aantal stations
614      */
615     public int getAantalStations() {
616         return stations.size();
617     }
618
619     /** Controleert op aanrijdingen met voorliggende taxi's.
620      * Als een taxi op de hoofdbaan zit dan worden voorliggende taxi's tot een bepaalde afstand gecontroleerd.
621      * Als een taxi op het station zit dan worden taxi's  op de tussenspoor en wisselsporen gecontroleerd.
622      * @param taxi Taxi waarvan voorliggende taxi's voor gecontroleerd moet worden.
623      * @return Is waar als een mogelijke aanrijding is gevonden en anders onwaar.
624      */
625     public boolean aanrijding(Taxi taxi) {
626         if (taxi.opSpoor()) {
627             if (taxi.getStatus() == TaxiStatus.RIJD || taxi.getStatus() == TaxiStatus.WACHT_OP_SPOOR) {
628                 if (taxi.opHoofdSpoor()) {
629                     double restAfstand = (taxi.getSpoorPositie() + MIN_AFSTAND_TOT_TAXI) - taxi.getSpoor().getLengte();
630                     if (restAfstand <= 0) {
631                         if (bevatTaxis(taxi.getSpoor(), taxi.getSpoorPositie(), taxi.getSpoor(), taxi.getSpoorPositie() + MIN_AFSTAND_TOT_TAXI)) return true;
632                     }
633                     else if (restAfstand > 0) {
634                         if (restAfstand < taxi.getSpoor().volgende().getLengte()) {
635                             if (bevatTaxis(taxi.getSpoor(), taxi.getSpoorPositie(), taxi.getSpoor().volgende(), restAfstand)) return true;
636                         }
637                         else
```

```
638                             if (bevatTaxis(taxi.getSpoor(), taxi.getSpoorPositie(), taxi.getSpoor().volgende(), taxi.getSpoor().volgende().getLengte())) return true;
639                         }
640                     }
641                 }
642             }
643             else if (taxi.opStation()) {
644                 Stationspoor sSpoor = taxi.getStation().getSpoor();
645                 if (bevatTaxis(sSpoor, sSpoor.getStation().getPositie(), sSpoor.volgende(), sSpoor.volgende().getLengte())) return true;
646                 if (bevatTaxis(sSpoor.vorige(), 0, sSpoor.volgende(), sSpoor.volgende().getLengte())) return true;
647             }
648             return false;
649         }
650 
651         /** Schakelt de eerstvolgende wisselspoor voor de opgegeven taxi. Schakelt de wisselspoor terug als een taxi de wisselspoor voorbij is.
652          * @param taxi De taxi waarvoor de wisselspoor (terug)geschakeld moet worden
653          * @return Is waar als de wisselspoor voor de taxi is (terug)geschakeld en onwaar als er niets is geschakeld
654          */
655         public boolean wisselaar(Taxi taxi) {
656             if (wisselLijst.containsKey(taxi)) {
657                 Wisselspoor wSpoor = wisselLijst.get(taxi);
658                 if (wSpoor.volgende().bevatTaxi(taxi)) {
659                     wSpoor.schakel();
660                     wisselLijst.remove(taxi);
661                     System.out.println("Wissel: " + wSpoor.getID() + " is teruggeschakeld door Taxi: " + taxi.getNaam());
662                     return true;
663                 }
664             }
665             else {
666                 Spoor spoor;
667                 if (taxi.opSpoor()) spoor = taxi.getSpoor();
668                 else spoor = taxi.getStation().getSpoor();
669                 if (spoor.volgende().getType() == SpoorType.WISSEL_SPOOR) {
670                     Wisselspoor wSpoor = (Wisselspoor) spoor.volgende();
671                     if (!wisselLijst.containsValue(wSpoor)) {
672                         wSpoor.schakel();
673                         wisselLijst.put(taxi, wSpoor);
674                         System.out.println("Wissel: " + wSpoor.getID() + " is geschakeld door Taxi: " + taxi.getNaam());
675                         return true;
676                     }
677                 }
678             }
679             return false;
680         }
681 
682         /** Controleert of taxi's op de tussenspoor van een station rijden of wachten.
683          * @param station Het station waarvan de tussenspoor op taxi's gecontroleerd moet worden.
684          * @return is waar als er taxi's op de tussenspoor rijden/wachten en anders onwaar.
685          */
686         public boolean taxisTussenSpoor(Station station) {
687             if (station.getSpoor().vorige().getNaSpoor().bevatTaxis()) return true;
688             else return false;
689         }
690 
691         /** Berekent afstand vanaf de positie van de taxi tot het station.
692          * @param t Taxi waar de afstand vanaf moet worden berekent.
693          * @param station Station waar de afstand tot moet worden berekent.
694          * @return afstand tussen de taxi en het station.
695          */
696         public long afstand(Taxi t, Station station) {
697             if (t.opStation()) {
698                 return bepaalAfstand(t.getStation(), station);
699             }
700             else if (t.opSpoor()) {
701                 long afstand = 0;
702                 boolean begin = false;
703                 Spoor taxiSpoor = t.getSpoor();
704                 for (Spoor s : sporen) {
705                     if (s == taxiSpoor) {
706                         afstand += s.getLengte() - t.getSpoorPositie();
707                         begin = true;
708                     }
```

```
709                 else if (begin) {
710                     if (s.getType() == SpoorType.WISSEL_SPOOR) {
711                         Wisselspoor wSpoor = (Wisselspoor) s;
712                         if (getStation(wSpoor) == station) {
713                             afstand += wSpoor.getLengte();
714                             afstand += wSpoor.getZijSpoor().getLengte() - ((Stationspoor)wSpoor.getZijSpoor()).getStation().getPositie();
715 
716                         }
717                     }
718                     else if (s.getType() == SpoorType.SPOOR) {
719                         afstand += s.getLengte();
720                     }
721                 }
722             }
723 
724         }
725         return 0;
726     }
727 
728     /** Berekent de afstand vanaf het begin van de baan tot aan de wisselspoor van het station.
729      * @param station Station waaraan de wisselspoor gekoppeld is.
730      * @return afstand tot het wisselspoor.
731      */
732     public long afstandTotZijSpoor(Station station) {
733         long afstand = 0;
734         for (Spoor s : sporen) {
735             if (s.getType() == SpoorType.WISSEL_SPOOR) {
736                 if (getStation((Wisselspoor)s) == station) break;
737                 else afstand += s.getLengte();
738             }
739             else if (s.getType() == SpoorType.SPOOR) afstand += s.getLengte();
740         }
741         return afstand;
742     }
743 
744     /** Geeft waar als taxi's vanaf de positie op het eerste spoor en tot de positie op het laatste spoor rijden of wachten en anders onwaar. Deze methode kan ook binnen één spoor controleren.
745      * @param s1 Eerste spoor
746      * @param van Beginpositie van de eerste spoor
747      * @param s2 Laatste spoor
748      * @param tot Eindpositie van de laatste spoor
749      * @return Is waar als een taxi is gevonden en anders onwaar.
750      */
751     public boolean bevatTaxis(Spoor s1, double van, Spoor s2, double tot) {
752         if (s1 == s2) {
753             for (Taxi t : s1.getTaxis())
754                 if (t.getSpoorPositie() > van && t.getSpoorPositie() < tot) return true;
755         }
756         else {
757             for (Spoor s : s1) {
758                 for (Taxi t : s.getTaxis()) {
759                     if (s == s1) { if (t.getSpoorPositie() > van) return true; }
760                     else if (s == s2) { if (t.getSpoorPositie() < tot) return true; }
761                     else return true;
762                 }
763                 if (s == s2) break;
764             }
765         }
766         return false;
767     }
768 
769     /** Geeft een station terug aan de hand van een wisselspoor.
770      * @param spoor Wisselspoor dat naar een station leidt.
771      * @return station waarnaar de opgegeven wisselspoor leidt.
772      */
773     public Station getStation(Wisselspoor spoor) {
774         return ((Stationspoor)spoor.getZijSpoor()).getStation();
775     }
776 
777     /** Geeft aan of er voorliggende taxi's op hetzelfde spoor als de opgegeven taxi zijn.
778      * @param taxi Taxi
779      * @return is waar als een voorliggende taxi is gevonden en anders onwaar
```

```
780         */
781        public boolean bevatTaxis(Taxi taxi) {
782            for (Taxi t : taxi.getSpoor().getTaxis())
783                if (t.getSpoorPositie() >= taxi.getSpoorPositie() && t != taxi) return true;
784            return false;
785        }
786
787        /** Geeft de wisselspoor die door de taxi is geschakeld.
788         * @param taxi Taxi dat een wissel heeft geschakeld
789         * @return de geschakelde wisselspoor
790         */
791        public Wisselspoor getGeschakeldeWisselspoor(Taxi taxi) {
792            return wisselLijst.get(taxi);
793        }
794
795        /** Geeft aan of een wisselspoor door de taxi is geschakeld.
796         * @param taxi - taxi
797         * @return Is waar als een wisselspoor door de taxi is geschakeld en anders onwaar.
798         */
799        public boolean isGeschakeldVoor(Taxi taxi) {
800            return wisselLijst.containsKey(taxi);
801        }
802
803        /** Geeft het eerstvolgende wisselspoor terug na de opgegeven spoor.
804         * @param huidigeSpoor Huidige spoor van een taxi.
805         * @return eerstvolgende wisselspoor
806         */
807        public Wisselspoor getVolgendeWisselspoor(Spoor huidigeSpoor) {
808            for (Spoor s : huidigeSpoor) {
809                if (s.getType() == SpoorType.WISSEL_SPOOR) return (Wisselspoor)s;
810            }
811            return null;
812        }
813
814        /** Geeft de tijd (in milliseconden) dat taxi's op een station moeten wachten.
815         * @return wachttijd van taxi's op een station.
816         */
817        public long getWachtTijdStation() {
818            return wachtTijdStation;
819        }
820
821        /** Stelt de tijd in dat taxi's op een station moeten wachten.
822         * @param wachtTijdStation Wachttijd (in milliseconden) op een station
823         */
824        public void setWachtTijdStation(int wachtTijdStation) {
825            this.wachtTijdStation = wachtTijdStation;
826        }
827
828        /** Geeft de snelheid (in meters per seconde) van taxi's
829         * @return snelheid van taxi's
830         */
831        public long getTaxiSnelheid() {
832            return taxiSnelheid;
833        }
834
835        /** Stelt de snelheid van taxi's in.
836         * @param taxiSnelheid Snelheid (in meters per seconde) van taxi's
837         */
838        public void setTaxiSnelheid(int taxiSnelheid) {
839            this.taxiSnelheid = taxiSnelheid;
840        }
841
842        /** Geeft de taxi met de opgegeven ID nummer terug.
843         * @param ID - ID nummer van een taxi
844         * @return taxi met de betreffende ID
845         * @throws NoSuchElementException Wanneer geen taxi met de opgegeven ID nummer is gevonden
846         */
847        public Taxi getTaxi(int ID) throws NoSuchElementException {
848            for (Taxi t : taxis) {
849                if (t.getID() == ID) return t;
850            }
```

```
851             throw new NoSuchElementException("Geen taxi met de ID: " + ID + " gevonden!");
852         }
853 
854         /** Geeft de station met de opgegeven naam terug.
855          * @param naam Naam van een taxi
856          * @return taxi met de betreffende naam.
857          * @throws NoSuchElementException wanneer geen taxi met de opgegeven naam is gevonden
858          */
859         public Taxi getTaxi(String naam) throws NoSuchElementException {
860             for (Taxi t : taxis) {
861                 if (t.getNaam().equals(naam)) return t;
862             }
863             throw new NoSuchElementException("Geen taxi met de naam: " + naam + " gevonden!");
864         }
865     }
866 
```

ReisManager.java

```
1     package org.tigam.railcab.algoritme;
2     import java.io.Serializable;
3     import java.util.ArrayList;
4     import java.util.Collections;
5     import java.util.EnumSet;
6     import java.util.LinkedList;
7     import java.util.NoSuchElementException;
8
9     /** De reismanager bevat de reizigers en reizen in de simulatie. De reismanager maakt (plant) reizen aan voor beschikbare reizigers en voert de reizen vervolgens uit. Reizigers kunnen via de <code>R
10     * @author Mustapha Bouzaidi
11     *
12     */
13    public class ReisManager implements Serializable, Runnable {
14        /**
15         *
16         */
17        private static final long serialVersionUID = -4919934401284854375L;
18        private BaanManager baanManager;
19        /**
20         * Maximaal aantal reizigers in een taxi.
21         */
22        public static final int MAX_REIZIGERS_IN_TAXI = 5;
23        /**
24         * Slaaptijd van de thread na het plannen van een reis en/of bijwerken van alle reizen.
25         */
26        public static final int THREAD_DELAY = 200;
27
28        public static final int NIET_INGESTELD = 0;
29        public static final int START = 1;
30        public static final int WACHT_OP_PAUZE = 2;
31        public static final int PAUZE = 4;
32        public static final int STOP = 8;
33        public static final int VERSNELD = 16;
34
35        private static int versnelFactor;
36        private double wachtTijden;
37        private int wachtTijdenAantal;
38        private int reizigerTeller;
39        private int status = NIET_INGESTELD;
40
41        private ArrayList<Reis> reizen;
42        private LinkedList<ReizigerWachtLijst> reizigerSet;
43
44        /** Creëert de reismanager met de opgegeven baanmanager.
45         * @param baanManager baanmanager dat door de reismanager wordt gebruikt
46         */
47        public ReisManager(BaanManager baanManager) {
48            this.baanManager = baanManager;
49            reizigerSet = new LinkedList<ReizigerWachtLijst>();
50            reizen = new ArrayList<Reis>();
51            status = NIET_INGESTELD;
52
53            versnelFactor = 1;
54            wachtTijden = 0;
55            wachtTijdenAantal = 0;
56            reizigerTeller = 0;
57        }
58
59        /** Geeft de status van de reismanager terug.
60         * @return Status van de reismanager
61         */
62        public synchronized int getStatus() {
63            return status;
64        }
65
66        /** Stelt de status van de reismanager in.
67         * @param status Status voor de reismanager
68         */
69        public synchronized void setStatus(int status) {
```

```
70          this.status = status;
71      }
72
73      /** Voert de reisplannings procedure uit. Vergelijkt maximaal MAX_REIZIGERS_IN_TAXI reizigers met de hoogste wachttijd en hetzelfde vertrekpunt/bestemmings- combinatie met de gemiddelde wachttij
74       * Vervolgens wordt het wachttijdpercentage van de eerdere vergelijking vergeleken met hoeveel taxi's er obezet zijn. Als het wachttijdpercentage hoger of gelijk is aan de taxibezetting dan word
75       * Een reis object wordt aangemaakt en vervolgens wordt de taxi genomen die het dichtst bij het vertrekpunt staat. Bovendien worden de reizigers aan de reis object toegevoegd.
76       *
77       */
78      public void planReis() {
79          if (!reizigerSet.isEmpty()) {
80              Collections.sort(reizigerSet);
81              ReizigerWachtLijst rWachtLijst = reizigerSet.getLast();
82              double cWachtTijden = 0;
83              int aantalReizigers = 0;
84              ArrayList<Taxi> beschikbareTaxis = baanManager.getBeschikbareTaxis();
85              if (!beschikbareTaxis.isEmpty() && !rWachtLijst.isEmpty()) {
86                  for (Reiziger r : rWachtLijst) {
87                      cWachtTijden += r.getWachtTijd();
88                      if (++aantalReizigers == MAX_REIZIGERS_IN_TAXI) {
89                          break;
90                      }
91                  }
92                  cWachtTijden /= aantalReizigers;
93                  double gWachtTijden = wachtTijden / wachtTijdenAantal;
94                  float aantalBeschikbareTaxis = beschikbareTaxis.size();
95                  float aantalTaxis = baanManager.getAantalTaxis();
96                  int wachtTijdPercentage = 0;
97                  if (wachtTijdenAantal != 0)
98                      wachtTijdPercentage = new Double(100 * (cWachtTijden / gWachtTijden)).intValue();
99                  else wachtTijdPercentage = 100;
100                 int taxiBezetting = new Float(100 * (1 - (aantalBeschikbareTaxis / aantalTaxis))).intValue();
101                 if (wachtTijdPercentage >= taxiBezetting) {
102                     Reis reis = new Reis();
103                     reis.setReisDetails(rWachtLijst.peek().getReisDetails());
104                     for (int i = 0; i < aantalReizigers; i++) {
105                         wachtTijden += rWachtLijst.peek().getWachtTijd();
106                         wachtTijdenAantal++;
107                         reis.voegReiziger(rWachtLijst.poll());
108                     }
109
110                     if (rWachtLijst.isEmpty()) reizigerSet.remove(rWachtLijst);
111
112                     Taxi taxi = null;
113                     Station vertrekpunt = reis.getReisDetails().getVertrekpunt();
114                     for (Taxi t : beschikbareTaxis) {
115                         if (taxi == null)
116                             taxi = t;
117                         else if (baanManager.afstand(taxi, vertrekpunt) > baanManager.afstand(t, vertrekpunt))
118                             taxi = t;
119                     }
120                     reis.setTaxi(taxi);
121                     taxi.setReis(reis);
122                     reizen.add(reis);
123                     reis.start();
124                 }
125             }
126         }
127     }
128
129     /**
130      * Werkt de wachttijd bij van alle reizigers zonder een reis.
131      */
132     public void updateOngeplandeReizigers() {
133         for (ReizigerWachtLijst rWachtLijst : reizigerSet) {
134             rWachtLijst.updateWachtTijden();
135         }
136     }
137
138     /** Pauzeert de reismanager door de thread in een wachtende loop te plaatsen en vervolgens de tijden van reizigers en reizen bij te werken.
139      * @return Is waar als de reismanager is gepauzeerd en anders onwaar
140      */
```

```
141     public boolean pauzeer() {
142         if (getStatus() != PAUZE || getStatus() != WACHT_OP_PAUZE) {
143             status = WACHT_OP_PAUZE;
144             while (getStatus() != ReisManager.PAUZE) {
145                 try {
146                     Thread.sleep(1);
147                 } catch (InterruptedException e) {
148                     e.printStackTrace();
149                 }
150             }
151             updateOngeplandeReizigers();
152             for (Reis reis : reizen) reis.pauzeer();
153             return true;
154         }
155         return false;
156     }
157 
158     /** Hervat de reismanager als het is gepauzeerd. De starttijden van reizigers en reizen worden gereset en vervolgens wordt de status aangepast.
159      * De thread van de reismanager wordt genotificeerd zodat het uit de pauze stand gaat.
160      * @return Is waar als de reismanager is hervat en anders onwaar.
161      */
162     public boolean hervat() {
163         if (status == PAUZE) {
164             for (ReizigerWachtLijst rWachtLijst : reizigerSet) {
165                 for (Reiziger r : rWachtLijst) {
166                     r.resetStartTijd();
167                 }
168             }
169             for (Reis reis : reizen) reis.hervat();
170             status = START;
171             return true;
172         }
173         return false;
174     }
175 
176     /** Geeft de baanmanager van de reismanager terug.
177      * @return BaanManager
178      */
179     public BaanManager getBaanManager() {
180         return baanManager;
181     }
182 
183     /** Stelt de baanmanager van de reismanager in.
184      * @param baanManager
185      */
186     public void setBaanManager(BaanManager baanManager) {
187         this.baanManager = baanManager;
188     }
189 
190     /** Geeft de reizen in de reismanager terug.
191      * @return reis objecten
192      */
193     public ArrayList<Reis> getReizen() {
194         return reizen;
195     }
196 
197     /** Stelt de reizen van de reismanager in.
198      * @param reizen reis objecten
199      */
200     public void setReizen(ArrayList<Reis> reizen) {
201         this.reizen = reizen;
202     }
203 
204     private ReizigerWachtLijst vindReizigers(ReisDetails rDetails) throws NoSuchElementException {
205         for (ReizigerWachtLijst rWachtLijst : reizigerSet) {
206             if (rWachtLijst.peek().getReisDetails().equals(rDetails)) return rWachtLijst;
207         }
208         throw new NoSuchElementException("Geen reizigers gevonden met de opgegeven vertrek- en bestemmingspunt.");
209     }
210 
211     /** Voegt een reiziger met het vertrekpunt en bestemming aan de reismanager toe. Gecontroleerd word of een reiziger aan een bestaande reis kan worden toegevoegd, anders wordt de reiziger in de l
```

```
212         * @param vertrekpunt Station
213         * @param bestemming Station
214         */
215        public synchronized void addReiziger(Station vertrekpunt, Station bestemming) {
216            reizigerTeller++;
217            Reiziger r = new Reiziger(reizigerTeller, "Reiziger " + reizigerTeller, ReizigerStatus.WACHT);
218            if (status != PAUZE) r.setStartTijd(System.currentTimeMillis());
219            ReisDetails rDetails = new ReisDetails(vertrekpunt, bestemming);
220            r.setReisDetails(rDetails);
221            if (!voegAanReis(r)) {
222                ReizigerWachtLijst rWachtLijst;
223                try {
224                    rWachtLijst = vindReizigers(rDetails);
225                } catch (NoSuchElementException e) {
226                    rWachtLijst = new ReizigerWachtLijst();
227                    reizigerSet.add(rWachtLijst);
228                }
229                vertrekpunt.voegReiziger(r);
230                rWachtLijst.add(r);
231            }
232        }
233
234        private boolean voegAanReis(Reiziger r) {
235            for (Reis reis : reizen) {
236                if (reis.getReisDetails().equals(r.getReisDetails())) {
237                    if (reis.naarVertrekpunt() && reis.getAantalReizigers() < MAX_REIZIGERS_IN_TAXI) {
238                        reis.voegReiziger(r);
239                        return true;
240                    }
241                }
242            }
243            return false;
244        }
245
246        @SuppressWarnings("unchecked")
247        public void run() {
248            while (status != STOP) {
249                synchronized (this) {
250                    while (status == WACHT_OP_PAUZE || status == PAUZE) {
251                        try {
252                            status = PAUZE;
253                            wait();
254                        } catch (InterruptedException e) {
255                            // TODO Auto-generated catch block
256                            e.printStackTrace();
257                        }
258                    }
259                }
260
261                updateOngeplandeReizigers();
262                planReis();
263
264                for (Reis reis : reizen) {
265                    Taxi taxi = reis.getTaxi();
266
267                    reis.updateReizigers();
268                    reis.updateReisTijd();
269
270                    if (baanManager.aanrijding(taxi)) {
271                        if (taxi.getStatus() == TaxiStatus.RIJD) {
272                            taxi.stop(TaxiReden.MOGELIJKE_AANRIJDING);
273                            System.out.println("Mogelijke aanrijding gedecteerd, " + taxi.getNaam() + " is gestopt!");
274                        }
275                    }
276
277                    else {
278
279                        if (taxi.getStatus() == TaxiStatus.WACHT_OP_SPOOR) {
280                            taxi.rij(TaxiReden.HERVAT_NA_MOGELIJKE_AANRIJDING);
281                            System.out.println(taxi.getNaam() + " hervat reis na mogelijke aanrijding.");
282                        }
```

```
283
284                     else if (taxi.opStation()) {
285                         Station station = taxi.getStation();
286                         taxi.updateWachtTijdOpStation();
287                         if (taxi.isOnbezet()) {
288                             if (station == reis.getReisDetails().getVertrekpunt()) {
289                                 reis.setStatus(EnumSet.of(ReisStatus.NAAR_VERTREKPUNT));
290                                 taxi.invoegen(station);
291                             } else {
292                                 reis.setStatus(EnumSet.of(ReisStatus.NAAR_VERTREKPUNT));
293                                 taxi.setWachtTijdOpStation(baanManager.getWachtTijdStation());
294                                 taxi.setStatus(TaxiStatus.WACHT_OP_STATION);
295                             }
296                         }
297
298                         if (station == reis.getReisDetails().getVertrekpunt()) {
299                             if (taxi.wachtOpReizigers()) {
300                                 for (Reiziger r : reis.getReizigers())
301                                     r.stapIn(taxi);
302                                 reis.setStatus(EnumSet.of(ReisStatus.NAAR_VERTREKPUNT, ReisStatus.REIZIGERS_OPGEHAALD));
303                             } else if (!taxi.wachtOpSluitingsTijd())
304                                 reis.setStatus(EnumSet.of(ReisStatus.NAAR_BESTEMMING, ReisStatus.REIZIGERS_OPGEHAALD));
305                         }
306
307                         else if (station == reis.getReisDetails().getBestemming() && reis.naarBestemming()) {
308                             reis.voltooien();
309                             Simulatie.getInstantie().voegReis(reis);
310                         }
311
312                         if (taxi.wachtOpUitrijden() && station.voorrang(taxi)) {
313                             if (baanManager.wisselaar(taxi) && baanManager.getGeschakeldeWisselspoor(taxi) == baanManager.getVolgendeWisselspoor(station.getSpoor()) && baanManager.getGeschakeldeWiss
314                                 taxi.rij(TaxiReden.UIT_STATION);
315                                 reis.resetStartUpdateTijd();
316                             }
317                         }
318                     }
319
320                     else if (taxi.opSpoor()) {
321                         if (!baanManager.isGeschakeldVoor(taxi)) {
322                             if (taxi.opHoofdSpoor() && (taxi.getSpoor().getLengte() - taxi.getSpoorPositie() <= BaanManager.MIN_AFSTAND_TOT_WISSEL)) {
323                                 Station station = baanManager.getStation(baanManager.getVolgendeWisselspoor(taxi.getSpoor()));
324                                 if (reis.naarVertrekpunt() && reis.getReisDetails().getVertrekpunt() == station || reis.naarBestemming() && reis.getReisDetails().getBestemming() == station)
325                                     if (!baanManager.bevatTaxis(taxi)) baanManager.wisselaar(taxi);
326                             }
327                         }
328                         else baanManager.wisselaar(taxi);
329                     }
330
331                     if (taxi.getStatus() == TaxiStatus.RIJD)
332                         taxi.updatePositie();
333                 }
334             }
335             for (int i = 0; i < reizen.size(); i++)
336                 if (reizen.get(i).reisVoltooid()) reizen.remove(i);
337
338             try {
339                 Thread.sleep(THREAD_DELAY);
340             } catch (InterruptedException e) {
341                 // TODO Auto-generated catch block
342                 e.printStackTrace();
343             }
344         }
345     }
346
347     /** Geeft de versnellingsfactor terug.
348      * @return versnellingsfactor in aantal maal versnelling
349      */
350     public static synchronized int getVersnelFactor() {
351         return versnelFactor;
352     }
353
```

```
354       /** Stelt de versnellingsfactor in.
355        * @param versnelFactor aantal maal versnelling
356        */
357       public static synchronized void setVersnelFactor(int versnelFactor) {
358           ReisManager.versnelFactor = versnelFactor;
359       }
360   }
```

Reis.java

```
1   package org.tigam.railcab.algoritme;
2   import java.io.Serializable;
3   import java.util.ArrayList;
4   import java.util.EnumSet;
5
6   /** De <code>Reis</code> klasse bevat reizigers en een taxi waarmee de reis wordt uitgevoerd. De <code>Reis</code> geeft aan waar een taxi moet gaan om reizigers op te halen en af te zetten. Bovendi
7    * @author Mustapha Bouzaidi
8    *
9    */
10  public class Reis implements Serializable {
11      /**
12       *
13       */
14      private static final long serialVersionUID = -1929403460392195371L;
15
16      private EnumSet<ReisStatus> status;
17
18      @SuppressWarnings("unused")
19      private long startUpdateTijd = 0;
20      @SuppressWarnings("unused")
21      private long wachtTijd = 0, reisTijdVertrekpunt = 0, reisTijdBestemming = 0;
22      private ArrayList<Reiziger> reizigers;
23      private Taxi taxi;
24      private ReisDetails reisDetails;
25
26      /**
27       * Creëert een reis zonder een taxi, reizigers en reisdetails (vertrekpunt en bestemming).
28       */
29      public Reis() {
30          this.reizigers = new ArrayList<Reiziger>();
31          status = EnumSet.noneOf(ReisStatus.class);
32      }
33
34      /** Creëert een reis met een taxi en reisdetails (vertrekpunt en bestemming).
35       * @param taxi De taxi die de reis uitvoert
36       * @param reisDetails het vertrekpunt en bestemming van de reis
37       */
38      public Reis(Taxi taxi, ReisDetails reisDetails) {
39          this.taxi = taxi;
40          this.reisDetails = reisDetails;
41          this.reizigers = new ArrayList<Reiziger>();
42          status = EnumSet.noneOf(ReisStatus.class);
43      }
44
45      /** Haalt een reiziger aan de hand van een index op.
46       * @param index positie in de lijst met reizigers
47       * @return reiziger met de opgegeven index
48       */
49      public Reiziger getReiziger(int index) {
50          return reizigers.get(index);
51      }
52
53      /** Voegt een reiziger toe aan de reis
54       * @param r reiziger om toe te voegen
55       */
56      public void voegReiziger(Reiziger r) {
57          reizigers.add(r);
58      }
59
60      /** Geeft de taxi terug die voor de reis wordt gebruikt.
61       * @return Taxi
62       */
63      public Taxi getTaxi() {
64          return taxi;
65      }
66
67      /** Stelt de taxi in die door de reis object wordt gebruikt.
68       * @param t Taxi
69       */
```

```
70        public void setTaxi(Taxi t) {
71            this.taxi = t;
72        }
73
74        /** Geeft de status van de taxi terug.
75         * @return status (combinaties mogelijk) van de reis
76         */
77        public EnumSet<ReisStatus> getStatus() {
78            return status;
79        }
80
81        /** Stelt de status van de reis in.
82         * @param status nieuwe status (combinaties mogelijk) van de reis object
83         */
84        public void setStatus(EnumSet<ReisStatus> status) {
85            this.status = status;
86        }
87
88        /** Geeft aan of de reizigers voor de reis zijn opgehaald.
89         * @return Is waar als de reizigers zijn opgehaald en anders onwaar
90         */
91        public boolean reizigersOpgehaald() {
92            return status.contains(ReisStatus.REIZIGERS_OPGEHAALD);
93        }
94
95        /** Geeft aan of de taxi voor de reis op weg is naar het vertrekpunt.
96         * @return Is waar als de taxi op weg is naar het vertrekpunt en anders onwaar
97         */
98        public boolean naarVertrekpunt() {
99            return status.contains(ReisStatus.NAAR_VERTREKPUNT);
100       }
101
102       /** Geeft aan of de taxi voor de reis op weg is naar de bestemming.
103        * @return Is waar als de taxi op weg is naar de bestemming en anders onwaar
104        */
105       public boolean naarBestemming() {
106           return status.contains(ReisStatus.NAAR_BESTEMMING);
107       }
108
109       /** Geeft aan of de reis is voltooid.
110        * @return Is waar als de reis is voltooid en anders onwaar
111        */
112       public boolean reisVoltooid() {
113           return status.contains(ReisStatus.VOLTOOID);
114       }
115
116       /** Geeft de totale reistijd van de reis terug.
117        * @return Reistijd in milliseconden
118        */
119       public long getReisTijd() {
120           return reisTijdVertrekpunt + reisTijdBestemming + wachtTijd;
121       }
122
123       /** Geeft de reistijd naar het vertrekpunt terug.
124        * @return Reistijd in milliseconden
125        */
126       public long getReisTijdVertrekpunt() {
127           return reisTijdVertrekpunt;
128       }
129
130       /** Stelt de reistijd naar het vertrekpunt in.
131        * @param tijd Reistijd in milliseconden
132        */
133       public void setReisTijdVertrekpunt(long tijd) {
134           this.reisTijdVertrekpunt = tijd;
135       }
136
137       /** Geeft de reistijd naar de bestemming terug.
138        * @return Reistijd in milliseconden
139        */
140       public long getReistijdBestemming() {
```

```
141         return reisTijdBestemming;
142     }
143 
144     /** Stelt de reistijd naar de bestemming in.
145      * @param tijd Reistijd in milliseconden
146      */
147     public void setReisTijdBestemming(long tijd) {
148         this.reisTijdBestemming = tijd;
149     }
150 
151     /** Geeft de reisdetails van de reis terug.
152      * @return reisdetails
153      */
154     public ReisDetails getReisDetails() {
155         return reisDetails;
156     }
157 
158     /** Stelt de reisdetails (vertrekpunt en bestemming) van de reis in.
159      * @param rDetails ReisDetails
160      */
161     public void setReisDetails(ReisDetails rDetails) {
162         this.reisDetails = rDetails;
163     }
164 
165     /**
166      * Print een bericht naar de console om aan te geven dat de reis is gestart.
167      */
168     public void start() {
169         System.out.println("Reis gestart met " + taxi.getNaam() + " en " + reizigers.size() + " reizigers.");
170     }
171 
172     /**
173      * Werkt de reistijd bij als de taxi tijdens de reis op weg is naar het vertrekpunt of bestemming.
174      */
175     public void updateReisTijd() {
176         if (taxi.opSpoor()) {
177             if (naarVertrekpunt())
178                 reisTijdVertrekpunt += (System.currentTimeMillis() - startUpdateTijd) * ReisManager.getVersnelFactor();
179             else if (naarBestemming())
180                 reisTijdBestemming += (System.currentTimeMillis() - startUpdateTijd) * ReisManager.getVersnelFactor();
181             startUpdateTijd = System.currentTimeMillis();
182         }
183     }
184 
185     /**
186      * Pauzeert de reis door de wachttijden en reistijden van de taxi en reizigers vooraf bij te werken en de taxi te stoppen.
187      */
188     public void pauzeer() {
189         if (taxi.opStation()) taxi.updateWachtTijdOpStation();
190         else if (taxi.opSpoor()) taxi.updatePositie();
191         taxi.stop(TaxiReden.PAUZE);
192         for (Reiziger r : reizigers) r.updateWachtTijd();
193         updateReisTijd();
194     }
195 
196     /**
197      * Hervat de reis door de starttijd voor het bijwerken van de reistijd en wachttijd op het station te resetten en de taxi weer te laten rijden als het op de spoor wacht.
198      */
199     public void hervat() {
200         resetStartUpdateTijd();
201         if (taxi.opStation()) taxi.resetStartTijdOpStation();
202         else if (taxi.opSpoor() && taxi.getStatus() == TaxiStatus.GEPAUZEERD_OP_SPOOR) taxi.rij(TaxiReden.HERVAT_NA_PAUZE);
203     }
204 
205     /**
206      * Starttijd voor het bijwerken van de reistijd wordt ingesteld op de huidige tijd van uitvoeren (reset).
207      */
208     public void resetStartUpdateTijd() {
209         startUpdateTijd = System.currentTimeMillis();
210     }
211 
```

```
212       /** Geeft het aantal reizigers van de reis terug.
213        * @return Het aantal reizigers
214        */
215       public int getAantalReizigers() {
216           return reizigers.size();
217       }
218
219       /** Geeft de reizigers van de reis terug.
220        * @return De reizigers
221        */
222       public ArrayList<Reiziger> getReizigers() {
223           return reizigers;
224       }
225
226       /**
227        * Voltooid de reis door de taxi uit de reis te halen en onbezet maken. Bovendien wordt de status van de reis op voltooid gezet.
228        */
229       public void voltooien() {
230           System.out.println("Reis voltooid met " + taxi.getNaam() + "!");
231           System.out.println("Reistijd naar vertrekpunt: " + ((double)reisTijdVertrekpunt / 1000) + " seconden.");
232           System.out.println("Reistijd naar bestemming: " + ((double)reisTijdBestemming / 1000) + " seconden");
233           System.out.println("Totale reisduur: " + ((double)getReisTijd() / 1000) + " seconden");
234           System.out.println("Wachttijden van reizigers:");
235           for (Reiziger r : reizigers) {
236               r.stapUit(taxi);
237               System.out.println("Wachttijd: " + ((double)r.getWachtTijd() / 1000) + " seconden");
238           }
239           taxi.setStatus(TaxiStatus.ONBEZET);
240           taxi.setReis(null);
241           this.setTaxi(null);
242           setStatus(EnumSet.of(ReisStatus.VOLTOOID));
243       }
244
245       /**
246        * Werkt de wachttijd van alle reizigers bij.
247        */
248       public void updateReizigers() {
249           if (reizigers.get(0).getStatus() == ReizigerStatus.WACHT) {
250               for (Reiziger r : reizigers) r.updateWachtTijd();
251           }
252       }
253
254       /** Stelt de wachttijd van de taxi tijdens de reis in.
255        * @param wachtTijd Wachttijd op het station in milliseconden
256        */
257       public void setWachtTijd(long wachtTijd) {
258           this.wachtTijd = wachtTijd;
259       }
260
261       /** Geeft de wachttijd van de taxi tijdens de reis terug.
262        * @return Wachttijd op het station in milliseconden
263        */
264       public long getWachtTijd() {
265           return wachtTijd;
266       }
267   }
268
```

ReisDetails.java

```
1   package org.tigam.railcab.algoritme;
2
3   import java.io.Serializable;
4
5   /**
6    * @author Mustapha Bouzaidi
7    *
8    */
9   public class ReisDetails implements Comparable<ReisDetails>, Serializable {
10      /**
11       *
12       */
13      private static final long serialVersionUID = -3414913389095080426L;
14      private Station vertrekpunt, bestemming;
15
16      /**
17       * Creëert een reisdetails object zonder een vertrekpunt en bestemming.
18       */
19      public ReisDetails() {
20      }
21
22      /** Creëert een reisdetails object met een vertrekpunt en bestemming.
23       * @param vertrekpunt Station waar de reizigers worden opgehaald
24       * @param bestemming Station waar de reizigers worden afgezet
25       */
26      public ReisDetails(Station vertrekpunt, Station bestemming) {
27          this.vertrekpunt = vertrekpunt;
28          this.bestemming = bestemming;
29      }
30
31      /** Geeft het vertrekpunt terug.
32       * @return Vertrekpunt station
33       */
34      public Station getVertrekpunt() {
35          return vertrekpunt;
36      }
37
38      /** Stelt het vertrekpunt in.
39       * @param vertrekpunt Station
40       */
```

```
41        public void setVertrekpunt(Station vertrekpunt) {
42            this.vertrekpunt = vertrekpunt;
43        }
44
45        /** Geeft de bestemming terug.
46         * @return Bestemming station
47         */
48        public Station getBestemming() {
49            return bestemming;
50        }
51
52        /** Stelt de bestemming in.
53         * @param bestemming Station
54         */
55        public void setBestemming(Station bestemming) {
56            this.bestemming = bestemming;
57        }
58
59        public boolean equals(Object o) {
60            ReisDetails r = (ReisDetails) o;
61            if (vertrekpunt.equals(r.vertrekpunt) && bestemming.equals(r.bestemming)) return true;
62            else return false;
63        }
64
65        public int compareTo(ReisDetails r) {
66            if (this.equals(r)) return 0;
67            else if (vertrekpunt.getID() > r.getVertrekpunt().getID()) return 1;
68            else if (vertrekpunt.getID() < r.getVertrekpunt().getID()) return -1;
69            else return -1;
70        }
71    }
72
```

ReisStatus.java

```
1     package org.tigam.railcab.algoritme;
2
3     import java.io.Serializable;
4
5     /**
6      * @author Mustapha Bouzaidi
7      *
8      */
9     public enum ReisStatus implements Serializable {
10        /**
11         * De taxi voor de reis is op weg naar het vertrekpunt.
12         */
13        NAAR_VERTREKPUNT,
14        /**
15         * De reizigers van de reis zijn opgehaald.
16         */
17        REIZIGERS_OPGEHAALD,
18        /**
19         * De taxi voor de reis is op weg naar de bestemming.
20         */
21        NAAR_BESTEMMING,
22        /**
23         * De reis is voltooid.
24         */
25        VOLTOOID;
26    }
27
```

Taxi.java

```
1   package org.tigam.railcab.algoritme;
2   import java.io.Serializable;
3   import java.util.ArrayList;
4   import java.util.Observable;
5   import java.util.Observer;
6
7   /** De <code>Taxi</code> is het vervoersmiddel in de simulatie waarmee reizigers van punt A naar B worden verplaatst d.m.v. een Reis. De taxi beweegt zich voort op spoordelen en voegt zich in als he
8    * @author Mustapha Bouzaidi
9    *
10   */
11  public class Taxi extends Observable implements Comparable<Taxi>, Serializable {
12      /**
13       *
14       */
15      private static final long serialVersionUID = -2416958873446309484L;
16
17      private int ID;
18
19      private String naam;
20
21      private TaxiStatus status;
22
23      private double absoluteAfstand, spoorAfstand;
24      private long startTijdStation, wachtTijdStation;
25
26      private long startUpdateTijdSpoor;
27
28      private Spoor spoor;
29      private ArrayList<Reiziger> reizigers;
30      private Reis reis;
31
32      private BaanManager baanManager;
33      private Station station;
34
35      /** Creëert een <code>Taxi</code> met een ID, naam en het <code>Station</code> waar het zal staan. Bovendien wordt de baanmanager gebruikt voor afstandsberekeningen.
36       * @param baanManager De te gebruiken baanmanager
37       * @param ID Identificatienummer van de taxi
38       * @param naam Naam van de taxi
39       * @param station Station waar de taxi staat.
40       */
41      public Taxi(BaanManager baanManager, int ID, String naam, Station station) {
42          this.baanManager = baanManager;
43          this.ID = ID;
44          this.naam = naam;
45          this.station = station;
46          this.absoluteAfstand = baanManager.afstandTotZijSpoor(station);
47          this.status = TaxiStatus.ONBEZET;
48          this.spoorAfstand = station.getPositie();
49          this.reizigers = new ArrayList<Reiziger>();
50      }
51
52      /** Creëert een <code>Taxi</code> met een ID, naam, een observer die op de hoogte gehouden wordt en het <code>Station</code> waar de taxi zal staan. Bovendien wordt de baanmanager gebruikt voor
53       * @param baanManager De te gebruiken baanmanager
54       * @param ID Identificatienummer van de taxi
55       * @param naam Naam van de taxi
56       * @param station Station waar de taxi staat.
57       * @param observer Wordt genotificeerd van wijzigingen aan de taxi.
58       */
59      public Taxi(BaanManager baanManager, int ID, String naam, Station station, Observer observer) {
60          this(baanManager, ID, naam, station);
61          this.addObserver(observer);
62      }
63
64      /** Stelt het <code>Station</code> in waarop deze taxi staat.
65       * @param station Station
66       */
67      public void setStation(Station station) {
68          this.station = station;
69      }
```

```
70
71      /** Geeft de identificatienummer van deze taxi terug.
72       * @return Identificatienummer
73       */
74      public int getID() {
75          return ID;
76      }
77
78      /** Stelt een nieuwe identificatienummer voor deze taxi in.
79       * @param ID Nieuwe identificatienummer
80       */
81      public void setID(int ID) {
82          this.ID = ID;
83      }
84
85      /** Geeft de naam terug van deze taxi.
86       * @return Naam van de taxi
87       */
88      public String getNaam() {
89          return naam;
90      }
91
92      /** Stelt een nieuwe naam in voor deze taxi.
93       * @param naam Nieuwe naam
94       */
95      public void setNaam(String naam) {
96          this.naam = naam;
97      }
98
99      /** Geeft de status van deze taxi terug.
100      * @return Status van de taxi
101      */
102     public TaxiStatus getStatus() {
103         return status;
104     }
105
106     /** Stelt een nieuwe status in voor deze taxi.
107      * @param status Nieuwe status
108      */
109     public void setStatus(TaxiStatus status) {
110         this.status = status;
111     }
112
113     /** Geeft de reis terug dat deze taxi uitvoert.
114      * @return De reis
115      * @throws RuntimeException Als deze taxi geen reis uitvoert.
116      */
117     public Reis getReis() throws RuntimeException {
118         if (reis == null) throw new RuntimeException("Taxi is op dit moment niet gekoppeld aan een reis.");
119         return reis;
120     }
121
122     /** Stelt een nieuwe <code>Reis</code> in dat deze taxi moet uitvoeren.
123      * @param r Nieuwe reis
124      */
125     public void setReis(Reis r) {
126         reis = r;
127     }
128
129     /** Laat de taxi op het spoor rijden. Als deze taxi op het station staat dan wordt het eerst uitgevoegd. Observer(s) worden genotificeerd met de opgegeven TaxiReden.
130      * @param reden Reden van rijden
131      * @return Is waar als deze taxi rijdt en anders onwaar
132      */
133     public boolean rij(TaxiReden reden) {
134         if (reden == TaxiReden.HERVAT_NA_MOGELIJKE_AANRIJDING || reden == TaxiReden.HERVAT_NA_PAUZE) {
135             startUpdateTijdSpoor = System.currentTimeMillis();
136             status = TaxiStatus.RIJD;
137             setChanged();
138             notifyObservers(reden);
139             clearChanged();
140             return true;
```

```
141          }
142          else if (reden == TaxiReden.UIT_STATION) {
143              uitvoegen();
144              startUpdateTijdSpoor = System.currentTimeMillis();
145              status = TaxiStatus.RIJD;
146              setChanged();
147              notifyObservers(reden);
148              clearChanged();
149              return true;
150          }
151          return false;
152      }
153
154      /** Laat deze taxi op het spoor of op het station stoppen.
155       * @param reden Reden van stoppen
156       * @return Is waar als deze taxi is gestopt en anders onwaar
157       */
158      public boolean stop(TaxiReden reden) {
159          if (getStatus() == TaxiStatus.RIJD) {
160              startUpdateTijdSpoor = 0;
161              if (reden == TaxiReden.MOGELIJKE_AANRIJDING || reden == TaxiReden.OP_STATION) status = TaxiStatus.WACHT_OP_SPOOR;
162              else if (reden == TaxiReden.PAUZE) status = TaxiStatus.GEPAUZEERD_OP_SPOOR;
163              setChanged();
164              notifyObservers(reden);
165              clearChanged();
166              return true;
167          }
168          return false;
169      }
170
171      /** Geeft de starttijd terug voor het bijwerken van de positie van deze taxi op het spoor.
172       * @return Starttijd voor het bijwerken van de spoorpositie
173       */
174      public long getStartUpdateTijdSpoor() {
175          return startUpdateTijdSpoor;
176      }
177
178      /** Stelt een nieuwe starttijd in voor het bijwerken van de positie van deze taxi op het spoor.
179       * @param tijd Nieuwe starttijd voor het bijwerken
180       */
181      public void setStartUpdateTijdSpoor(long tijd) {
182          this.startUpdateTijdSpoor = tijd;
183      }
184
185      /** Geeft de absolute positie (positie op de hoofdbaan vanuit het startpunt) van deze taxi.
186       * @return absolute positie
187       */
188      public double getAbsolutePositie() {
189          return absoluteAfstand;
190      }
191
192      /** Geeft de positie van deze taxi op de zijbaan terug.
193       * De zijbaan is een combinatie van het stationspoor en de bijbehorende wisselsporen.
194       * Als de taxi niet richting het stationspoor of uit het station gaat dan is de positie op de zijbaan gelijk aan 0.
195       * @return Positie op de zijbaan
196       */
197      public double getZijPositie() {
198          if (opSpoor()) {
199              if (spoor.getType() == SpoorType.WISSEL_SPOOR) {
200                  Wisselspoor wSpoor = (Wisselspoor) spoor;
201                  if (wSpoor.getRichting() == Wisselspoor.RICHTING_VOOR_ZIJSPOOR && wSpoor.isGeschakeld())
202                      return spoorAfstand;
203                  else if (wSpoor.getRichting() == Wisselspoor.RICHTING_NA_ZIJSPOOR && wSpoor.isGeschakeld())
204                      return spoor.vorige().vorige().getLengte() + spoor.vorige().getLengte() + spoorAfstand;
205              } else if (spoor.getType() == SpoorType.STATION_SPOOR) {
206                  return spoor.vorige().getLengte() + spoorAfstand;
207              }
208          } else if (opStation()) {
209              return station.getZijPositieIn();
210          }
211          return 0;
```

```
212     }
213
214     /** Werkt de spoor positie en absolute positie van deze taxi bij. Als deze taxi een station bereikt dan wordt het ingevoegd.
215      *
216      */
217     public void updatePositie() {
218         if (status == TaxiStatus.RIJD) {
219             if (opStationSpoor()) {
220                 Stationspoor sSpoor = (Stationspoor) spoor;
221                 Station huidigeStation = sSpoor.getStation();
222                 if ((huidigeStation == reis.getReisDetails().getVertrekpunt() && !reis.reizigersOpgehaald()) || (huidigeStation == reis.getReisDetails().getBestemming() && reis.reizigersOpgehaald())
223                     if (spoorAfstand >= sSpoor.getStation().getPositie()) {
224                         stop(TaxiReden.OP_STATION);
225                         invoegen(huidigeStation);
226                         spoorAfstand = sSpoor.getStation().getPositie();
227                         System.out.println(naam + " zit op: " + huidigeStation.getNaam());
228                     }
229                 }
230             }
231
232             if (opSpoor()) {
233                 double verstrekenTijd = ((System.currentTimeMillis() - startUpdateTijdSpoor) / 1000.0);
234                 double afgelegdeAfstand = verstrekenTijd * (baanManager.getTaxiSnelheid() * ReisManager.getVersnelFactor());
235
236                 if ((spoorAfstand + afgelegdeAfstand) >= spoor.getLengte()) {
237                     if (spoor.getType() == SpoorType.SPOOR || (spoor.getType() == SpoorType.WISSEL_SPOOR && !((Wisselspoor)spoor).isGeschakeld())) {
238                         absoluteAfstand += spoor.getLengte() - spoorAfstand;
239                     }
240                     else if (spoor.getType() == SpoorType.WISSEL_SPOOR && ((Wisselspoor)spoor).getRichting() == Wisselspoor.RICHTING_NA_ZIJSPOOR) {
241                         absoluteAfstand += spoor.getLengte();
242                         absoluteAfstand += spoor.getVoorSpoor().getLengte();
243                         absoluteAfstand += spoor.getVoorSpoor().getVoorSpoor().getLengte();
244                     }
245                     double restAfstand = (spoorAfstand + afgelegdeAfstand) -  spoor.getLengte();
246                     spoorAfstand = 0;
247                     setSpoor(spoor.volgende());
248                     while (restAfstand >= spoor.getLengte()) {
249                         restAfstand -= spoor.getLengte();
250
251                         if (spoor.getType() == SpoorType.SPOOR || (spoor.getType() == SpoorType.WISSEL_SPOOR && !((Wisselspoor)spoor).isGeschakeld())) {
252                             if (spoor == baanManager.getSporen()) absoluteAfstand = 0;
253                             absoluteAfstand += spoor.getLengte();
254                         }
255                         else if (spoor.getType() == SpoorType.WISSEL_SPOOR && ((Wisselspoor)spoor).getRichting() == Wisselspoor.RICHTING_NA_ZIJSPOOR) {
256                             absoluteAfstand += spoor.getLengte();
257                             absoluteAfstand += spoor.getVoorSpoor().getLengte();
258                             absoluteAfstand += spoor.getVoorSpoor().getVoorSpoor().getLengte();
259                         }
260                         setSpoor(spoor.volgende());
261                     }
262                     if (restAfstand > 0) {
263                         if (spoor.getType() == SpoorType.SPOOR || (spoor.getType() == SpoorType.WISSEL_SPOOR && !((Wisselspoor)spoor).isGeschakeld())) {
264                             if (spoor == baanManager.getSporen()) absoluteAfstand = 0;
265                             absoluteAfstand += restAfstand;
266                         }
267                         spoorAfstand += restAfstand;
268                     }
269                 } else {
270                     if (spoor.getType() == SpoorType.SPOOR || (spoor.getType() == SpoorType.WISSEL_SPOOR && !((Wisselspoor)spoor).isGeschakeld())) {
271                         absoluteAfstand += afgelegdeAfstand;
272                     }
273                     spoorAfstand += afgelegdeAfstand;
274                 }
275                 startUpdateTijdSpoor = System.currentTimeMillis();
276             }
277             if (opStationSpoor() && naarStation() && (getSpoorPositie() > ((Stationspoor)spoor).getStation().getPositie()))
278                 updatePositie();
279             else {
280                 if (opSpoor()) System.out.format("%s zit op: %s, SpoorAfstand: %.2f en AbsoluteAfstand: %.2f en ZijAfstand: %.2f. Versnelfactor: %d.%n", getNaam(), getSpoor().getID(), spoorAfstand,
281                 setChanged();
282                 notifyObservers();
```

```
283                 clearChanged();
284             }
285         }
286     }
287
288     /** Geeft aan of deze taxi op het station is.
289      * @return Is waar als deze taxi op het station is en anders onwaar
290      */
291     public boolean opStation() {
292         if ((status == TaxiStatus.WACHT_OP_STATION || status == TaxiStatus.ONBEZET ||status == TaxiStatus.GEPAUZEERD_OP_SPOOR) && station != null) return true;
293         return false;
294     }
295
296     /** Geeft aan of deze taxi op de zijbaan naar het station rijdt.
297      * @return Is waar als deze taxi naar het station rijdt en anders onwaar
298      */
299     public boolean naarStation() {
300         if (opSpoor()) {
301             if (spoor.getType() == SpoorType.WISSEL_SPOOR) {
302                 Wisselspoor wSpoor = (Wisselspoor) spoor;
303                 if (wSpoor.getRichting() == Wisselspoor.RICHTING_VOOR_ZIJSPOOR && wSpoor.isGeschakeld())
304                     return true;
305             }
306             else if (spoor.getType() == SpoorType.STATION_SPOOR) {
307                 Stationspoor sSpoor = (Stationspoor) spoor;
308                 if (sSpoor.getStation() == getReis().getReisDetails().getVertrekpunt() && !getReis().reizigersOpgehaald())
309                     return true;
310                 else if (sSpoor.getStation() == getReis().getReisDetails().getBestemming() && getReis().reizigersOpgehaald() && getReis().naarBestemming())
311                     return true;
312                 else if (getSpoorPositie() <= sSpoor.getStation().getPositie()) return true;
313             }
314         }
315         return false;
316     }
317
318     /** Geeft aan of deze taxi op de zijbaan vanuit het station rijdt.
319      * @return Is waar als deze taxi uit het station en zijbaan rijdt en anders onwaar
320      */
321     public boolean vanStation() {
322         if (opSpoor()) {
323             if (spoor.getType() == SpoorType.WISSEL_SPOOR) {
324                 Wisselspoor wSpoor = (Wisselspoor) spoor;
325                 if (wSpoor.getRichting() == Wisselspoor.RICHTING_NA_ZIJSPOOR && wSpoor.isGeschakeld())
326                     return true;
327             }
328             else if (spoor.getType() == SpoorType.STATION_SPOOR) {
329                 Stationspoor sSpoor = (Stationspoor) spoor;
330                 if (sSpoor.getStation() == getReis().getReisDetails().getVertrekpunt() && getReis().reizigersOpgehaald())
331                     return true;
332                 else if (sSpoor.getStation() == getReis().getReisDetails().getBestemming() && getSpoorPositie() > sSpoor.getStation().getPositie() && !getReis().naarBestemming())
333                     return true;
334                 else if (getSpoorPositie() > sSpoor.getStation().getPositie())
335                     return true;
336             }
337         }
338         return false;
339     }
340
341     /** Geeft aan of deze taxi op een spoor staat.
342      * @return Is waar als deze taxi op een spoor staat en anders onwaar
343      */
344     public boolean opSpoor() {
345         if (spoor != null && (status == TaxiStatus.WACHT_OP_SPOOR || status == TaxiStatus.RIJD || status == TaxiStatus.GEPAUZEERD_OP_SPOOR)) return true;
346         return false;
347     }
348
349     /** Geeft aan of deze taxi op een stationspoor staat.
350      * @return Is waar als deze taxi op een stationspoor staat en anders onwaar
351      */
352     public boolean opStationSpoor() {
353         if (spoor != null && spoor.getType() == SpoorType.STATION_SPOOR) return true;
```

```
354         else return false;
355     }
356 
357     /** Geeft aan of deze taxi op het hoofdspoor staat.
358      * @return Is waar als deze taxi op het hoofdspoor staat en anders onwaar
359      */
360     public boolean opHoofdSpoor() {
361         if (spoor != null && spoor.getType() == SpoorType.SPOOR) return true;
362         else return false;
363     }
364 
365     /** Geeft het spoor waar deze taxi op staat terug.
366      * @return Het spoor waar deze taxi op staat
367      * @throws RuntimeException Wanneer de taxi niet op een spoor staat
368      */
369     public Spoor getSpoor() throws RuntimeException {
370         if (opSpoor()) return spoor;
371         throw new RuntimeException("De taxi staat niet op een spoor");
372     }
373 
374     /** Stelt een nieuwe spoor voor deze taxi in en voegt de taxi aan het spoordeel toe.
375      * @param spoor Nieuwe spoor
376      */
377     public void setSpoor(Spoor spoor) {
378         if (this.spoor != null) this.spoor.verwijderTaxi(this);
379         this.spoor = spoor;
380         if (spoor != null) spoor.voegTaxi(this);
381     }
382 
383     /** Geeft de positie terug op het huidige spoordeel waar deze taxi op staat.
384      * @return Positie op het huidige spoordeel
385      */
386     public double getSpoorPositie() {
387         return spoorAfstand;
388     }
389 
390     /** Geeft het station waar deze taxi op staat terug.
391      * @return Het station
392      * @throws RuntimeException Wanneer deze taxi niet op een station staat.
393      */
394     public Station getStation() throws RuntimeException {
395         if (opStation()) return station;
396         throw new RuntimeException("De taxi staat niet op het station");
397     }
398 
399     /** Geeft de wachttijd van deze taxi op het station terug.
400      * @return wachttijd op het station
401      */
402     public long getWachtTijdOpStation() {
403         return wachtTijdStation;
404     }
405 
406     /** Stelt een nieuwe starttijd in voor het bijwerken van de wachttijd van deze taxi op het station.
407      * @param startTijd Nieuwe starttijd
408      */
409     public void setStartTijdOpStation(long startTijd) {
410         this.startTijdStation = startTijd;
411     }
412 
413     /** Stelt een nieuwe positie in voor deze taxi op het huidige spoordeel.
414      * @param positie Positie op het spoor
415      */
416     public void setSpoorPositie(long positie) {
417         spoorAfstand = positie;
418     }
419 
420     /** Vergelijkt 2 Taxi's aan de hand van de ID, naam, spoordeel, station, reizigers en reis van de taxi's.
421      * @param t Taxi om mee te vergelijken
422      * @return Is waar als de 2 taxi's gelijk zijn aan elkaar en anders onwaar
423      */
424     public boolean equals(Taxi t) {
```

```
425         if (getID() != t.getID()) return false;
426         if (!getNaam().equals(t.getNaam())) return false;
427         if (opSpoor() != t.opSpoor()) return false;
428         if (opStation() != t.opStation()) return false;
429         if (getReizigers() != t.getReizigers()) return false;
430         if (getReis() != t.getReis()) return false;
431         return true;
432     }
433 
434     public int compareTo(Taxi t) {
435         if (!isOnbezet() && !t.isOnbezet()) {
436             if (getWachtTijdOpStation() > t.getWachtTijdOpStation()) return 1;
437             else if (getWachtTijdOpStation() < t.getWachtTijdOpStation()) return -1;
438             else return 0;
439         }
440         else if (isOnbezet() && !t.isOnbezet()) return -1;
441         else if (getWachtTijdOpStation() > t.getWachtTijdOpStation()) return 1;
442         else if (getWachtTijdOpStation() < t.getWachtTijdOpStation()) return -1;
443         return 0;
444     }
445 
446     /** Voegt de reiziger toe aan deze taxi.
447      * @param reiziger Reiziger om toe te voegen
448      * @return Is waar als de reiziger is toegevoegd en anders onwaar
449      */
450     public boolean voegReiziger(Reiziger reiziger) {
451         return reizigers.add(reiziger);
452     }
453 
454     /** Verwijdert de reiziger uit deze taxi.
455      * @param reiziger Reiziger om te verwijderen
456      * @return Is waar als de reiziger is verwijderd en anders onwaar
457      */
458     public boolean verwijderReiziger(Reiziger reiziger) {
459         return reizigers.remove(reiziger);
460     }
461 
462     /** Geeft aan of deze taxi de opgegeven reiziger bevat.
463      * @param reiziger De reiziger
464      * @return Is waar als de reiziger in de taxi zit/staat en anders onwaar
465      */
466     public boolean bevatReiziger(Reiziger reiziger) {
467         return reizigers.contains(reiziger);
468     }
469 
470     /** Geeft de reizigers in deze taxi terug.
471      * @return Lijst met reizigers
472      */
473     public ArrayList<Reiziger> getReizigers() {
474         return reizigers;
475     }
476 
477     /** Geeft het aantal reizigers in deze taxi terug.
478      * @return Het aantal reizigers
479      */
480     public int getAantalReizigers() {
481         return reizigers.size();
482     }
483 
484     /** Geeft aan of deze taxi op het station op de reizigers van zijn reis wacht.
485      * @return Is waar als deze taxi op de reizigers van zijn reis wacht en anders onwaar.
486      */
487     public boolean wachtOpReizigers() {
488         if (opStation() && getStation() == getReis().getReisDetails().getVertrekpunt() && !getReis().reizigersOpgehaald()) return true;
489         return false;
490     }
491 
492     /** Geeft aan of deze taxi op het station wacht totdat het de deuren kan sluiten (wachttijd verstreken).
493      * @return Is waar als de wachttijd op het station is verstreken en anders onwaar
494      */
495     public boolean wachtOpSluitingsTijd() {
```

```
496         if (opStation() && getWachtTijdOpStation() < baanManager.getWachtTijdStation()) return true;
497         return false;
498     }
499 
500     /** Geeft aan of deze taxi wacht voordat het uit het station kan rijden.
501      * @return Is waar als deze taxi wacht op het uitrijden en anders onwaar
502      */
503     public boolean wachtOpUitrijden() {
504         if (opStation() && getWachtTijdOpStation() >= baanManager.getWachtTijdStation()) {
505             if (getStation() != reis.getReisDetails().getVertrekpunt()) return true;
506             else if (reis.reizigersOpgehaald()) return true;
507         }
508         return false;
509     }
510 
511     /** Geeft aan of deze taxi onbezet en op een station staat.
512      * @return Is waar als deze taxi onbezet is en anders onwaar
513      */
514     public boolean isOnbezet() {
515         if (opStation() && getStatus() == TaxiStatus.ONBEZET) return true;
516         return false;
517     }
518 
519     /** Voert het uitvoegingsprocedure van deze taxi bij een station uit.
520      * Deze taxi rijdt uit het station en op het spoor.
521      *
522      */
523     public void uitvoegen() {
524         getStation().voegTaxiUit(this);
525         setSpoorPositie(getStation().getPositie());
526         setSpoor(getStation().getSpoor());
527         setStation(null);
528         setStartTijdOpStation(0);
529         setWachtTijdOpStation(0);
530     }
531 
532     /** Werkt de wachttijd van deze taxi op het station bij zolang het niet onbezet is.
533      *
534      */
535     public void updateWachtTijdOpStation() {
536         if (opStation() && !isOnbezet()) {
537             long verstrekenTijd = (System.currentTimeMillis() - startTijdStation) * ReisManager.getVersnelFactor();
538             wachtTijdStation += verstrekenTijd;
539             reis.setWachtTijd(reis.getWachtTijd() + verstrekenTijd);
540             resetStartTijdOpStation();
541         }
542     }
543 
544     /** Voert het invoegingsprocedure van deze taxi als het op een station is aangekomen. Deze taxi rijdt het station binnen en uit het spoor.
545      * @param s Station om in te voegen
546      */
547     public void invoegen(Station s) {
548         setStartTijdOpStation(System.currentTimeMillis());
549         setWachtTijdOpStation(0);
550         if (getStatus() != TaxiStatus.ONBEZET) s.voegTaxiIn(this);
551         setStation(s);
552         setSpoor(null);
553         setStatus(TaxiStatus.WACHT_OP_STATION);
554     }
555 
556     /** Geeft de starttijd terug voor het bijwerken van de wachttijd op het station.
557      * @return Starttijd voor het bijwerken van de wachttijd
558      */
559     public long getStartTijdOpStation() {
560         return startTijdStation;
561     }
562 
563     /**
564      * Stelt de starttijd van deze taxi (voor het bijwerken van de wachttijd op het station) in op de huidige tijd (reset).
565      */
566     public void resetStartTijdOpStation() {
```

```
567             startTijdStation = System.currentTimeMillis();
568         }
569 
570         /** Stelt een nieuwe wachttijd voor deze taxi op het station in.
571          * @param tijd Nieuwe
572          */
573         public void setWachtTijdOpStation(long tijd) {
574             wachtTijdStation = tijd;
575         }
576 
577         /** Geeft de namen van de reizigers in deze taxi terug.
578          * @return Namen van de reizigers in deze taxi
579          */
580         public String[] getReizigerNamen() {
581             String[] namen = new String[reizigers.size()];
582             for (int i = 0; i < reizigers.size(); i++) {
583                 namen[i] = reizigers.get(i).getNaam();
584             }
585             return namen;
586         }
587 
588         /** Geeft de reiziger in deze taxi met de opgegeven naam terug.
589          * @param naam Naam van de reiziger
590          * @return Reiziger in deze taxi met de opgegeven naam
591          * @throws RuntimeException Wanneer de reiziger niet in deze taxi is gevonden
592          */
593         public Reiziger getReiziger(String naam) throws RuntimeException {
594             for (Reiziger r : reizigers) {
595                 if (r.getNaam().equals(naam)) return r;
596             }
597             throw new RuntimeException("Reiziger is niet in de taxi gevonden");
598         }
599 
600         /** Geeft aan of de reiziger met de opgegeven naam in deze taxi zit.
601          * @param naam Naam van de reiziger
602          * @return Is waar als de reiziger in deze taxi zit en anders onwaar
603          */
604         public boolean bevatReiziger(String naam) {
605             for (Reiziger r : reizigers) {
606                 if (r.getNaam().equals(naam)) return true;
607             }
608             return false;
609         }
610     }
611 
```

TaxiStatus.java

```
1     package org.tigam.railcab.algoritme;
2
3     import java.io.Serializable;
4
5     /** De status van een <code>Taxi</code>.
6      * @author Mustapha Bouzaidi
7      *
8      */
9     public enum TaxiStatus implements Serializable {
10        /**
11         * De <code>Taxi</code> rijdt op het <code>Spoor</code>.
12         */
13        RIJD,
14        /**
15         * De <code>Taxi</code> wacht/is gestopt op het <code>Spoor</code>.
16         */
17        WACHT_OP_SPOOR,
18        /**
19         * De <code>Taxi</code> wacht op het <code>Station</code>.
20         */
21        WACHT_OP_STATION,
22        /**
23         * De <code>Taxi</code> is onbezet op het <code>Station</code>.
24         */
25        ONBEZET,
26        /**
27         * De <code>Taxi</code> is bezet.
28         */
29        BEZET,
30        /**
31         * De <code>Taxi</code> staat gepauzeerd op het <code>Spoor</code>.
32         */
33        GEPAUZEERD_OP_SPOOR;
34    }
35
```

TaxiReden.java

```
1     package org.tigam.railcab.algoritme;
2
3     import java.io.Serializable;
4
5     /** De <code>TaxiReden</code> enumeratie wordt door de <code>Taxi</code> klasse gebruikt om de reden van een handeling (zoals het stoppen of rijden van een taxi) aan te geven. Bovendien wordt <code>
6      * @author Mustapha Bouzaidi
7      *
8      */
9     public enum TaxiReden implements Serializable {
10        /**
11         * De <code>Taxi</code> stopt vanwege mogelijke aanrijding tussen taxi's.
12         */
13        MOGELIJKE_AANRIJDING,
14        /**
15         * De <code>Taxi</code> rijdt verder na een mogelijke aanrijding
16         */
17        HERVAT_NA_MOGELIJKE_AANRIJDING,
18        /**
19         * De <code>Taxi</code> rijdt het <code>Station</code> uit.
20         */
21        UIT_STATION,
22        /**
23         * De <code>Taxi</code> stopt op het <code>Station</code>.
24         */
25        OP_STATION,
26        /**
27         * De <code>Simulatie</code> is gepauzeerd.
28         */
29        PAUZE,
30        /**
31         * De <code>Simulatie</code> is hervat.
32         */
33        HERVAT_NA_PAUZE;
34    }
35
```

Reiziger.java

```
1     package org.tigam.railcab.algoritme;
2
3     import java.io.Serializable;
4
5     /** De reiziger klasse representeert een reiziger in de simulatie die vanaf een vertrekpunt naar een bestemming wil reizen met een <code>Taxi</code>. De reiziger houdt zijn wachttijd bij als het nog
6      * @author Mustapha Bouzaidi
7      *
8      */
9     public class Reiziger implements Comparable<Reiziger>, Serializable {
10        /**
11         *
12         */
13        private static final long serialVersionUID = -6096969574405515181L;
14        private int ID;
15        private String naam;
16
17        private ReizigerStatus status;
18        private long startTijd = 0, wachtTijd = 0;
19        private ReisDetails reisDetails;
20
21        /** Creëert een reiziger met een ID, naam en status.
22         * @param ID Identificatie nummer
23         * @param naam Naam van de reiziger
24         * @param status Status van de reiziger
25         */
26        public Reiziger(int ID, String naam, ReizigerStatus status) {
27            this.ID = ID;
28            this.naam = naam;
29            this.status = status;
30        }
31
32        /** Creëert een reiziger met een ID, nam, starttijd en status.
33         * @param ID Identificatie nummer
34         * @param naam Naam van de reiziger
35         * @param startTijd starttijd voor het bijwerken van de wachttijd
36         * @param status Status van de reiziger
37         */
38        public Reiziger(int ID, String naam, long startTijd, ReizigerStatus status) {
39            this(ID, naam, status);
40            this.startTijd = startTijd;
41        }
42
43        /** Geeft het identificatie nummer van de reiziger terug.
44         * @return Identificatie nummer
45         */
46        public int getID() {
47            return ID;
48        }
49
50        /** Stelt de identificatie nummer van de reiziger in.
51         * @param ID Nieuwe identificatie nummer
52         */
53        public void setID(int ID) {
54            this.ID = ID;
55        }
56
57        /** Geeft de naam van de reiziger terug.
58         * @return Naam van de reiziger
59         */
60        public String getNaam() {
61            return naam;
62        }
63
64        /** Stelt de naam van de reiziger in.
65         * @param naam Naam voor de reiziger
66         */
67        public void setNaam(String naam) {
68            this.naam = naam;
69        }
```

```
70
71      /** Geeft de wachttijd van de reiziger terug.
72       * @return Wachttijd voor de reiziger
73       */
74      public long getWachtTijd() {
75          return wachtTijd;
76      }
77
78      /** Stelt de wachttijd van de reiziger in.
79       * @param tijd Nieuwe wachttijd van de reiziger
80       */
81      public void setWachtTijd(long tijd) {
82          this.wachtTijd = tijd;
83      }
84
85      /** Geeft de status van de reiziger terug.
86       * @return Status van de reiziger
87       */
88      public ReizigerStatus getStatus() {
89          return status;
90      }
91
92      /** Stelt de status van de reiziger in.
93       * @param status Nieuwe status van de reiziger
94       */
95      public void setStatus(ReizigerStatus status) {
96          this.status = status;
97      }
98
99      /** Voegt de reiziger uit het station en in de taxi. De status van de reiziger veranderd naar ReisStatus.REIST.
100      * @param taxi Taxi waar de reiziger in moet stappen
101      * @return Is waar als de reiziger in de taxi is gestapt en anders onwaar
102      */
103     public boolean stapIn(Taxi taxi) {
104         if (taxi.voegReiziger(this)) {
105             taxi.getStation().verwijderReiziger(this);
106             status = ReizigerStatus.REIST;
107             return true;
108         }
109         return false;
110     }
111
112     /** Voegt de reiziger uit de taxi. Verandert de status van de reiziger naar ReisStatus.AANGEKOMEN.
113      * @param taxi Taxi waar de reiziger uit moet stappen.
114      */
115     public void stapUit(Taxi taxi) {
116         if (taxi.verwijderReiziger(this)) {
117             status = ReizigerStatus.AANGEKOMEN;
118         }
119     }
120
121     /** Geeft de reisdetails van de reiziger terug.
122      * @return Reisdetails van de reiziger
123      */
124     public ReisDetails getReisDetails() {
125         return reisDetails;
126     }
127
128     /** stelt de reisdetails van de reiziger in.
129      * @param rDetails Nieuwe reisdetails
130      */
131     public void setReisDetails(ReisDetails rDetails) {
132         this.reisDetails = rDetails;
133     }
134
135     /** Vergelijkt 2 reizigers aan de hand van het identificatienummer, naam, status, wachttijd en reisdetails.
136      * @param r Reiziger om mee te vergelijken
137      * @return Is waar als de 2 reizigers gelijk en anders onwaar.
138      */
139     public boolean equals(Reiziger r) {
140         if (ID != r.getID() && !naam.equals(r.getNaam())) return false;
```

```
141             if (!status.equals(r.getStatus()) && getWachtTijd() != r.getWachtTijd()) return false;
142             if (!reisDetails.equals(r.getReisDetails())) return false;
143             return true;
144         }
145 
146         public int compareTo(Reiziger r) {
147             if (equals(r) || getStartTijd() == r.getStartTijd()) return 0;
148             else if (getStartTijd() > r.getStartTijd()) return 1;
149             else if (getStartTijd() < r.getStartTijd()) return -1;
150             else return 0;
151         }
152 
153         /** Geeft de starttijd (voor het bijwerken van de wachttijd) van de reiziger terug.
154          * @return Starttijd van de reiziger
155          */
156         public long getStartTijd() {
157             return startTijd;
158         }
159 
160         /** Stelt de starttijd van de reiziger in
161          * @param startTijd Begintijd waarmee wachttijd wordt bijgewerkt.
162          */
163         public void setStartTijd(long startTijd) {
164             this.startTijd = startTijd;
165         }
166 
167         /** Stelt de starttijd van de reiziger in op de huidige tijd in milliseconden (System.currentTimeMillis).
168          *
169          */
170         public void resetStartTijd() {
171             this.startTijd = System.currentTimeMillis();
172         }
173 
174         /** Geeft aan of de starttijd van de reiziger is ingesteld (hoger dan de waarde 0).
175          * @return Is waar als de starttijd van de reiziger is ingesteld en anders onwaar.
176          */
177         public boolean startTijdIsSet() {
178             if (startTijd == 0) return false;
179             else return true;
180         }
181 
182         /**
183          * Werkt de wachttijd bij zolang de reiziger de status ReizigerStatus.WACHT heeft. Starttijd wordt gereset na het bijwerken van de wachttijd.
184          */
185         public void updateWachtTijd() {
186             if (status == ReizigerStatus.WACHT) {
187                 if (startTijdIsSet()) {
188                     wachtTijd += (System.currentTimeMillis() - startTijd) * ReisManager.getVersnelFactor();
189                     resetStartTijd();
190                 }
191             }
192         }
193     }
```

ReizigerStatus.java

```
1     package org.tigam.railcab.algoritme;
2
3     import java.io.Serializable;
4
5     /**
6      * @author Mustapha Bouzaidi
7      *
8      */
9     public enum ReizigerStatus implements Serializable {
10        /**
11         * Reiziger wacht op een taxi.
12         */
13        WACHT,
14        /**
15         * Reiziger zit of reist in een taxi.
16         */
17        REIST,
18        /**
19         * Reiziger is aangekomen op de bestemming.
20         */
21        AANGEKOMEN;
22    }
23
```

ReizigerWachtLijst.java

```
1   package org.tigam.railcab.algoritme;
2
3   import java.io.Serializable;
4   import java.util.Iterator;
5   import java.util.concurrent.PriorityBlockingQueue;
6
7   /**
8    * @author Mustapha Bouzaidi
9    *
10   */
11  public class ReizigerWachtLijst implements Comparable<ReizigerWachtLijst>, Iterable<Reiziger>, Serializable {
12      /**
13       *
14       */
15      private static final long serialVersionUID = 44534085006565521 85L;
16      private PriorityBlockingQueue<Reiziger> reizigers;
17      private long wachtTijden;
18
19      /**
20       * Creëert een wachtlijst voor reizigers met hetzelfde vertrekpunt en bestemming.
21       */
22      public ReizigerWachtLijst() {
23          reizigers = new PriorityBlockingQueue<Reiziger>();
24          wachtTijden = 0;
25      }
26
27      /** Voegt een reiziger toe aan de wachtlijst.
28       * @param r Reiziger om toe te voegen
29       * @return Is waar als de reiziger is teogevoegd en anders onwaar.
30       */
31      public boolean add(Reiziger r) {
32          return reizigers.add(r);
33      }
34
35      /** Geeft de grootte van de wachtlijst terug.
36       * @return grootte van de wachtlijst
37       */
38      public int size() {
39          return reizigers.size();
40      }
```

```
41
42          public Iterator<Reiziger> iterator() {
43              return reizigers.iterator();
44          }
45
46          /** Neemt een kijkje van de laatste reiziger in de wachtlijst.
47           * @return Laatste reiziger in de wachtlijst
48           */
49          public Reiziger peek() {
50              return reizigers.peek();
51          }
52
53          /** Haalt de laatste reiziger uit de wachtlijst.
54           * @return Laatste reiziger uit de wachtlijst
55           */
56          public Reiziger poll() {
57              return reizigers.poll();
58          }
59
60          /** Geeft aan of de wachtlijst leeg is.
61           * @return Is waar als de wachtlijst leeg is en anders onwaar.
62           */
63          public boolean isEmpty() {
64              return reizigers.isEmpty();
65          }
66
67          /**
68           * Werkt de wachttijden van alle reizigers bij in de wachtlijst.
69           * Bovendien wordt de cumulatieve wachttijd in de Reizigerwachtlijst ook bijgewerkt.
70           */
71          public void updateWachtTijden() {
72              wachtTijden = 0;
73              for (Reiziger r : reizigers) {
74                  r.updateWachtTijd();
75                  wachtTijden += r.getWachtTijd();
76              }
77          }
78
79          /** Geeft de cumulatieve wachttijd van de reizigers in de wachtlijst terug.
80           * @return Cumulatieve wachttijd
81           */
82          public long getWachtTijden() {
```

```
83              return wachtTijden;
84          }
85
86          public int compareTo(ReizigerWachtLijst rLijst) {
87              if (isEmpty())
88                  if (rLijst.isEmpty()) return 0;
89                  else return -1;
90              else if (rLijst.isEmpty()) return 1;
91
92              if (getWachtTijden() == rLijst.getWachtTijden()) return 0;
93              if (getWachtTijden() > rLijst.getWachtTijden()) return 1;
94              if (getWachtTijden() < rLijst.getWachtTijden()) return -1;
95              else return 0;
96          }
97
98      }
```

Station.java

```
1   package org.tigam.railcab.algoritme;
2   import java.io.Serializable;
3   import java.util.ArrayList;
4   import java.util.Collections;
5   import java.util.LinkedList;
6   import java.util.Observable;
7
8   /** Het <code>station</code> is een onderdeel van de baanstructuur. Stations bevatten reizigers en/of taxi's.
9    * @author Mustapha Bouzaidi
10   *
11   */
12  public class Station extends Observable implements Serializable {
13      /**
14       *
15       */
16      private static final long serialVersionUID = 5958689306036998509L;
17
18      private int ID;
19      private String naam;
20
21      private ArrayList<Reiziger> reizigers;
22      private Stationspoor spoor;
23      private long positie;
24
25      private LinkedList<Taxi> taxis;
26
27      /** Creëert een Station met een ID, naam en een positie op een <code>Stationspoor</code>.
28       * @param ID Identificatienummer van het station
29       * @param naam Naam van het station
30       * @param positieOpSpoor Positie van het station op een <code>Stationspoor</code>
31       */
32      public Station(int ID, String naam, long positieOpSpoor) {
33          this.ID = ID;
34          this.naam = naam;
35          this.positie = positieOpSpoor;
36          reizigers = new ArrayList<Reiziger>();
37          taxis = new LinkedList<Taxi>();
38      }
39
40      /** Creëert een Station met een ID, naam, <code>Stationspoor</code> en een positie op het <code>Stationspoor</code>.
41       * @param ID Identificatienummer van het station
42       * @param naam Naam van het station
43       * @param spoor Stationspoor waar het station op zit.
44       * @param positieOpSpoor Positie van het station op een <code>Stationspoor</code>
45       */
46      public Station(int ID, String naam, Stationspoor spoor, long positieOpSpoor) {
47          this(ID, naam, positieOpSpoor);
48          this.spoor = spoor;
49          this.spoor.setStation(this);
50      }
51
52      /** Geeft de reizigers op dit <code>Station</code> terug.
53       * @return Lijst met reizigers
```

```
54          */
55         public ArrayList<Reiziger> getReizigers() {
56             return reizigers;
57         }
58 
59         /** Geeft de namen van de reizigers op dit <code>Station</code> terug.
60          * @return Namen van de reizigers
61          */
62         public String[] getReizigerNamen() {
63             String[] namen = new String[reizigers.size()];
64             for (int i = 0; i < reizigers.size(); i++) {
65                 namen[i] = reizigers.get(i).getNaam();
66             }
67             return namen;
68         }
69 
70         /** Geeft de reiziger met de opgegeven naam terug.
71          * @param naam Naam van de reiziger
72          * @return Reiziger met de opgegeven naam.
73          */
74         public Reiziger getReiziger(String naam) {
75             for (Reiziger r : reizigers) {
76                 if (r.getNaam().equals(naam)) return r;
77             }
78             throw new RuntimeException("Reiziger is niet op het station gevonden");
79         }
80 
81         /** Geeft het aantal reizigers op dit <code>Station</code> terug.
82          * @return Het aantal reizigers
83          */
84         public int getAantalReizigers() {
85             return reizigers.size();
86         }
87 
88         /** Geeft het aantal taxi's op dit <code>Station</code> terug.
89          * @return Het aantal taxi's
90          */
91         public int getAantalTaxis() {
92             return taxis.size();
93         }
94 
95         /** Geeft het identificatienummer van dit <code>Station</code> terug.
96          * @return Identificatienummer
97          */
98         public int getID() {
99             return ID;
100        }
101 
102        /** Stelt het identificatienummer van dit <code>Station</code> in.
103         * @param ID Nieuwe identificatienummer
104         */
105        public void setID(int ID) {
106            this.ID = ID;
107        }
108 
```

```
109       /** Geeft de naam van dit <code>Station</code> terug.
110        * @return Naam van het <code>Station</code>
111        */
112       public String getNaam() {
113           return naam;
114       }
115
116       /** Stelt de naam van dit <code>Station</code> in.
117        * @param naam Nieuwe naam voor het <code>Station</code>.
118        */
119       public void setNaam(String naam) {
120           this.naam = naam;
121       }
122
123       /** Voegt de taxi aan dit <code>Station</code> toe.
124        * @param t Taxi om toe te voegen
125        * @return Is waar als de taxi aan het <code>Station</code> is toegevoegd en anders onwaar
126        */
127       public boolean voegTaxiIn(Taxi t) {
128           return taxis.add(t);
129       }
130
131       /** Verwijdert de taxi uit dit <code>Station</code>.
132        * @param t Taxi om te verwijderen
133        * @return Is waar als de taxi is verwijderd en anders onwaar
134        */
135       public boolean voegTaxiUit(Taxi t) {
136           return taxis.remove(t);
137       }
138
139       /** Geeft aan of de taxi voorrang heeft om uit dit <code>Station</code> te rijden. De taxi met de hoogste wachttijd heeft voorrang om uit te rijden.
140        * @param t Taxi dat wil uitrijden
141        * @return Is waar als de taxi voorrang heeft en anders onwaar.
142        */
143       public boolean voorrang(Taxi t) {
144           Collections.sort(taxis);
145           if (taxis.getLast().equals(t) && !t.isOnbezet()) return true;
146           return false;
147       }
148
149       /** Geeft het <code>Stationspoor</code> terug waar dit <code>Station</code> op staat.
150        * @return Stationspoor
151        */
152       public Stationspoor getSpoor() {
153           return spoor;
154       }
155
156       /** Geeft de positie op het <code>Stationspoor</code> terug.
157        * @return Positie
158        */
159       public long getPositie() {
160           return positie;
161       }
162
163       /** Geeft de positie terug van het <code>Station</code> vanuit het begin van de zijbaan.
```

```
164         * @return Positie
165         */
166        public long getZijPositieIn() {
167            return spoor.vorige().getLengte() + positie;
168        }
169
170        /** Geeft de positie terug van het <code>Station</code> tot aan het einde van de zijbaan.
171         * @return Positie
172         */
173        public long getZijPositieUit() {
174            return spoor.volgende().getLengte() + (spoor.getLengte() - positie);
175        }
176
177        /** Stelt de positie van het <code>Station</code> op het <code>Stationspoor</code> in.
178         * @param positieOpSpoor
179         */
180        public void setPositie(long positieOpSpoor) {
181            this.positie = positieOpSpoor;
182        }
183
184        /** Stelt het nieuwe <code>Stationspoor</code> in waar dit <code>Station</code> op zit.
185         * @param spoor Nieuwe <code>Stationspoor</code>
186         */
187        public void setSpoor(Stationspoor spoor) {
188            this.spoor = spoor;
189        }
190
191        /** Verwijdert de reiziger uit dit <code>Station</code>.
192         * @param r Reiziger om te verwijderen
193         * @return Is waar als de reiziger is verwijderd uit dit <code>Station</code> en anders onwaar.
194         */
195        public boolean verwijderReiziger(Reiziger r) {
196            return reizigers.remove(r);
197        }
198
199        /** Voegt de reizigers aan dit <code>Station</code>.
200         * @param r Reiziger om toe te voegen.
201         * @return Is waar als de reiziger is toegevoegd aan dit <code>Station</code> en anders onwaar.
202         */
203        public boolean voegReiziger(Reiziger r) {
204            return reizigers.add(r);
205        }
206
207    }
208
```

Spoor.java

```
1     package org.tigam.railcab.algoritme;
2     import java.io.Serializable;
3     import java.util.ArrayList;
4
5     /** Het <code>Spoor</code> is een onderdeel van de baanstructuur en dient vaak als een hoofdspoordeel.
6      * Taxi's kunnen zich op een <code>Spoor</code> voortbewegen.
7      * <code>Spoor</code> objecten zijn d.m.v. een dubbel linked list implementatie aan elkaar gekoppeld.
8      * @author Mustapha Bouzaidi
9      *
10     */
11    public class Spoor implements Iterable<Spoor>, Serializable {
12        /**
13         *
14         */
15        private static final long serialVersionUID = 581013334049799349 1L;
16        /**
17         * Alphanumerieke identficatienummer.
18         */
19        protected String ID;
20        /**
21         * Lengte van het spoordeel.
22         */
23        protected long lengte;
24        /**
25         * Het soort spoordeel, zie Spoortype enumeratie.
26         */
27        protected SpoorType type;
28        /**
29         * Een spoordeel dat gekoppeld is aan de voorzijde van dit spoordeel.
30         */
31        protected Spoor voorSpoor;
32        /**
33         * Een spoordeel dat gekoppeld is aan de achterzijde van dit spoordeel.
34         */
35        protected Spoor naSpoor;
36        /**
37         * De taxi(s) die zich op dit spoordeel bevinden.
38         */
39        protected ArrayList<Taxi> taxis;
40
41        /** Creëert een spoordeel met een identificatie nummer en een spoorlengte.
42         * @param id Alphanumerieke identificatienummer
43         * @param l Lengte van het spoordeel
44         */
45        public Spoor(String id, long l) {
```

```
46          ID = id;
47          lengte = l;
48          type = SpoorType.SPOOR;
49          taxis = new ArrayList<Taxi>();
50      }
51
52      /** Creëert een spoordeel met een identificatie nummer en een spoorlengte.
53       * @param id Alphanumerieke identificatienummer
54       * @param l Lengte van het spoordeel
55       * @param voor Het spoordeel dat voor dit spoordeel gekoppeld is.
56       * @param na Het spoordeel dat na dit spoordeel volgt
57       */
58      public Spoor(String id, long l, Spoor voor, Spoor na) {
59          this(id, l);
60          setVoorSpoor(voor);
61          setNaSpoor(na);
62      }
63
64      /** Geeft de lengte van het spoordeel terug.
65       * @return Lengte van het spoordeel
66       */
67      public long getLengte() {
68          return lengte;
69      }
70
71      /** Stelt de lengte van het spoordeel in.
72       * @param lengte Nieuwe lengte van het spoordeel
73       */
74      public void setLengte(long lengte) {
75          this.lengte = lengte;
76      }
77
78      /** Geeft het SpoorType van dit spoordeel terug. Verschillende soorten spoordelen hebben variërende methoden.
79       * @return Type van het spoor
80       */
81      public SpoorType getType() {
82          return type;
83      }
84
85      /** Stelt het soort spoor van dit spoordeel in.
86       * @param type Type van het spoor
87       */
88      public void setType(SpoorType type) {
89          this.type = type;
90      }
91
92      /** Geeft het volgende spoordeel terug waaraan dit spoordeel is verbonden.
93       * @return Volgende spoordeel
```

```
94          */
95         public Spoor getNaSpoor() {
96             return naSpoor;
97         }
98 
99         /** geeft het volgende spoordeel terug maar dit kan bij een <code>Wisselspoor</code> variëren omdat het geschakeld kan worden.
100         * @return Volgende spoordeel
101         */
102        public Spoor volgende() {
103            return naSpoor;
104        }
105 
106        /** Geeft het vorige spoordeel terug maar dit kan bij een <code>Wisselspoor</code> variëren omdat het geschakeld kan worden.
107         * @return Vorige spoordeel
108         */
109        public Spoor vorige() {
110            return voorSpoor;
111        }
112 
113        /** Geeft de taxi met de opgegeven index terug.
114         * @param index positie in de lijst met taxi's
115         * @return De taxi met de opgegeven index in de lijst.
116         */
117        public Taxi getTaxi(int index) {
118            return taxis.get(index);
119        }
120 
121        /** Geeft aan of de opgegeven taxi zich op dit spoordeel bevindt.
122         * @param taxi De taxi
123         * @return Is waar als de taxi zich op dit spoordeel bevindt en anders onwaar
124         */
125        public boolean bevatTaxi(Taxi taxi) {
126            return taxis.contains(taxi);
127        }
128 
129        /** Geeft aan hoeveel taxi's zich op dit spoordeel bevinden.
130         * @return Het aantal taxi's
131         */
132        public int aantalTaxis() {
133            return taxis.size();
134        }
135 
136        /** Voegt de opgegeven taxi aan dit spoordeel toe.
137         * @param taxi De toe te voegen taxi.
138         */
139        public void voegTaxi(Taxi taxi) {
140            this.taxis.add(taxi);
141        }
```

```
142
143       /** Verwijdert de opgegeven taxi uit dit spoordeel.
144        * @param taxi De te verwijderen taxi
145        */
146       public void verwijderTaxi(Taxi taxi) {
147           this.taxis.remove(taxi);
148       }
149
150       /** Geeft aan of taxi(s) zich wel of niet bevinden op dit spoordeel.
151        * @return Is waar als taxi(s) zich op het spoordeel bevinden en anders onwaar.
152        */
153       public boolean bevatTaxis() {
154           return !taxis.isEmpty();
155       }
156
157       public SpoorIterator iterator() {
158           return new SpoorIterator(this);
159       }
160
161       /** Geeft de taxi's op dit spoordeel terug.
162        * @return Lijst met taxi(s)
163        */
164       public ArrayList<Taxi> getTaxis() {
165           return taxis;
166       }
167
168       /** Geeft het alphanumerieke identificatienummer van dit spoordeel terug
169        * @return Identificatienummer
170        */
171       public String getID() {
172           return ID;
173       }
174
175       /** Stelt het alphanumerieke identificatienummer voor dit spoordeel in.
176        * @param ID Identificatienummer
177        */
178       public void setID(String ID) {
179           this.ID = ID;
180       }
181
182       /** Geeft het vorige spoordeel terug waaraan dit spoordeel verbonden is.
183        * @return Vorige spoordeel
184        */
185       public Spoor getVoorSpoor() {
186           return voorSpoor;
187       }
188
189       /** Stelt het vorige spoordeel in waaraan dit spoordeel is verbonden.
```

```
190         * @param spoor Vorige spoordeel
191         */
192        public void setVoorSpoor(Spoor spoor) {
193            this.voorSpoor = spoor;
194        }
195
196        /** Stelt het volgende spoordeel in waaraan dit spoordeel is verbonden.
197         * @param spoor Volgende spoordeel
198         */
199        public void setNaSpoor(Spoor spoor) {
200            spoor.setVoorSpoor(this);
201            this.naSpoor = spoor;
202        }
203    }
204
```

Wisselspoor.java

```
1   package org.tigam.railcab.algoritme;
2   /**
3    * @author Mustapha Bouzaidi
4    *
5    */
6   public class Wisselspoor extends Spoor {
7       /**
8        *
9        */
10      private static final long serialVersionUID = 2320308939392597123L;
11      /**
12       * Geeft aan dat het <code>Wisselspoor</code> is gekoppeld aan het hoofdspoor.
13       */
14      public static final boolean STAND_HOOFD_SPOOR = false;
15      /**
16       * Geeft aan dat het <code>Wisselspoor</code> is geschakeld en gekoppeld aan het zijspoor.
17       */
18      public static final boolean STAND_ZIJ_SPOOR = true;
19      /**
20       * Geeft aan dat het <code>Wisselspoor</code> voor het rijden naar de zijbaan gebruikt wordt.
21       */
22      public static final boolean RICHTING_VOOR_ZIJSPOOR = false;
23      /**
24       * Geeft aan dat het <code>Wisselspoor</code> voor het rijden uit de zijbaan gebruikt wordt.
25       */
26      public static final boolean RICHTING_NA_ZIJSPOOR = true;
27
28      private boolean stand;
29      private boolean richting;
30      private Spoor zijSpoor;
31
32      /** Creëert een <code>Wisselspoor</code> met een ID, lengte en een richting.
33       * @param ID Alphanumerieke Identificatienummer
34       * @param lengte Lengte van het wisselspoor
35       * @param richting Richting voor of na de zijbaan.
36       */
37      public Wisselspoor(String ID, long lengte, boolean richting) {
38          super(ID, lengte);
39          stand = STAND_HOOFD_SPOOR;
40          this.richting = richting;
41          type = SpoorType.WISSEL_SPOOR;
42      }
43
44      /** Creëert een <code>Wisselspoor</code> met een ID, lengte, richting en een voor- na- en zijspoor.
45       * @param ID Alphanumerieke Identificatienummer
46       * @param lengte Lengte van het wisselspoor
47       * @param voor Vorige spoordeel
48       * @param na Volgende spoordeel
49       * @param zij Zijspoordeel
50       * @param richting Richting voor of na de zijbaan.
51       */
52      public Wisselspoor(String ID, long lengte, Spoor voor, Spoor na, Spoor zij, boolean richting) {
53          super(ID, lengte, voor, na);
54          this.richting = richting;
55          setZijSpoor(zij);
56          stand = STAND_HOOFD_SPOOR;
57          type = SpoorType.WISSEL_SPOOR;
58      }
```

```
59
60          /** Stelt in op welke stand dit wisselspoor moet staan.
61           * @param stand Stand van het wisselspoor
62           */
63          public void setStand(boolean stand) {
64              this.stand = stand;
65          }
66
67          /**
68           * Schakelt dit wisselspoor in of schakelt het terug als het al is geschakelt.
69           */
70          public void schakel() {
71              stand = !stand;
72          }
73
74          /** Geeft aan of het wisselspoor is geschakeld.
75           * @return Is waar als het wisselspoor naar het zijspoor wijst en onwaar als het naar het hoofdspoor wijst.
76           */
77          public boolean isGeschakeld() {
78              return stand;
79          }
80
81          /** Geeft het zijspoor terug waarnaar dit wisselspoor verwijst.
82           * @return Zijspoor
83           */
84          public Spoor getZijSpoor() {
85              return zijSpoor;
86          }
87
88          /** Stelt het zijspoor in waarnaar dit wisselspoor moet verwijzen.
89           * @param spoor Zijspoor
90           */
91          public void setZijSpoor(Spoor spoor) {
92              if (richting == RICHTING_VOOR_ZIJSPOOR) spoor.setVoorSpoor(this);
93              else spoor.setNaSpoor(this);
94              this.zijSpoor = spoor;
95          }
96
97          public Spoor volgende() {
98              if (richting == RICHTING_VOOR_ZIJSPOOR)
99                  if (stand) return zijSpoor;
100                 else return naSpoor;
101             else return naSpoor;
102         }
103
104         public Spoor vorige() {
105             if (richting == RICHTING_NA_ZIJSPOOR)
106                 if (stand) return zijSpoor;
107                 else return voorSpoor;
108             else return voorSpoor;
109         }
110
111         /** Geeft aan of dit <code>Wisselspoor</code> voor het rijden naar- of voor het rijden uit- de zijbaan bedoeld is.
112          * @return Is waar als dit <code>Wisselspoor</code> voor het rijden uit de zijbaan gebruikt wordt en onwaar als het voor het inrijden in de zijbaan gebruikt wordt.
113          */
114         public boolean getRichting() {
115             return richting;
116         }
117     }
118
```

Stationspoor.java

```
1     package org.tigam.railcab.algoritme;
2
3     /**
4      * @author Mustapha Bouzaidi
5      *
6      */
7     public class Stationspoor extends Spoor {
8         /**
9          *
10         */
11        private static final long serialVersionUID = -4274766648729085085L;
12        private Station station;
13
14        /** Creëert een <code>Stationspoor</code> met een ID, spoorlengte en het <code>Station</code> dat erop staat.
15         * @param ID Alphanumerieke identificatienummer
16         * @param lengte Lengte van dit spoordeel
17         * @param station <code>Station</code> dat op dit <code>Stationspoor</code> staat.
18         */
19        public Stationspoor(String ID, long lengte, Station station) {
20            super(ID, lengte);
21            this.station = station;
22            this.station.setSpoor(this);
23            type = SpoorType.STATION_SPOOR;
24        }
25
26        /** Creëert een <code>Stationspoor</code> met een ID, spoorlengte en het <code>Station</code> dat erop staat.
27         * Bovendien wordt het vorige en volgende spoor van dit spoordeel gekoppeld.
28         * @param ID Alphanumerieke identificatienummer
29         * @param lengte Lengte van dit spoordeel
30         * @param voor Vorige spoordeel
31         * @param na Volgende spoordeel
32         * @param station <code>Station</code> dat op dit <code>Stationspoor</code> staat.
33         */
34        public Stationspoor(String ID, long lengte, Spoor voor, Spoor na, Station station) {
35            super(ID, lengte, voor, na);
36            this.station = station;
37            this.station.setSpoor(this);
38            type = SpoorType.STATION_SPOOR;
39        }
40
```

```
41        /** Geeft het <code>Station</code> terug dat op dit <code>Stationspoor</code> staat.
42         * @return Het <code>Station</code> op dit <code>Stationspoor</code>.
43         */
44        public Station getStation() {
45            return station;
46        }
47
48        /** Stelt het nieuwe <code>Station</code> in dat op dit <code>Stationspoor</code> moet staan.
49         * @param station Nieuwe station
50         */
51        public void setStation(Station station) {
52            this.station = station;
53        }
54    }
55
```

SpoorType.java

```
1     package org.tigam.railcab.algoritme;
2
3     import java.io.Serializable;
4
5     /** De <code>SpoorType</code> enumeratie geeft aan wat voor Spoordeel een instantie van de (super)klasse <code>Spoor</code> heeft.
6      * @author Mustapha Bouzaidi
7      *
8      */
9     public enum SpoorType implements Serializable {
10        /**
11         * Geeft aan dat het om een normale <code>Spoor</code> gaat, meestal is dit het hoofdspoor.
12         */
13        SPOOR,
14        /**
15         * Geeft aan dat een het om een <code>Wisselspoor</code> gaat.
16         */
17        WISSEL_SPOOR,
18        /**
19         * Geeft aan dat het om een <code>Stationspoor</code> gaat.
20         */
21        STATION_SPOOR;
22    }
23
```

SpoorIterator.java

```
1     package org.tigam.railcab.algoritme;
2     import java.util.Iterator;
3
4     /** Een Iterator voor baanstructuren. Itereert over verschillende spoordelen uit de baanstructuur en maakt hierbij gebruik van de gezamenlijke methode getNaSpoor(). Door het gebruik van getNaSpoor w
5      * @author Mustapha
6      *
7      */
8     public class SpoorIterator implements Iterator<Spoor> {
9         private Spoor sporen;
10
11        /** Creëert een <code>SpoorIterator</code> voor de opgegeven baanstructuur.
12         * @param spoor Baanstructuur (ofwel begin spoordeel van de baanstructuur)
13         */
14        public SpoorIterator(Spoor spoor) {
15            this.sporen = spoor;
16        }
17
18        public boolean hasNext() {
19            if (sporen.getNaSpoor() != null) return true;
20            else return false;
21        }
22
23        public Spoor next() {
24            Spoor spoor = sporen;
25            sporen = sporen.getNaSpoor();
26            return spoor;
27        }
28
29        public void remove() {
30            throw new UnsupportedOperationException();
31        }
32    }
33
```