1 CyclingPortal.java

```
1  package cycling;
2
3  import java.util.Arrays;
4
5  import java.io.IOException;
6  import java.time.LocalDateTime;
7  import java.time.LocalTime;
8  import java.util.ArrayList;
9  import java.io.ObjectOutputStream;
10 import java.io.FileOutputStream;
11 import java.io.ObjectInputStream;
12 import java.io.FileInputStream;
13
14
15
16 /**
17  * CyclingPortal implements CyclingPortalInterface; contains
        methods for
18  * handling the following classes: Race, Stage, Segment,
        RiderManager (and in
19  * turn Rider and Team), and Result.
20  * These classes are used manage races and their subdivisions,
        teams and their
21  * riders, and to calculate and assign points.
22  * Also contains methods for saving and loading
        MiniCyclingPortalInterface to
23  * and from a file.
24  *
25  * @author Ethan Ray & Thomas Newbold
26  * @version 1.0
27  *
28  */
29 public class CyclingPortal implements CyclingPortalInterface {
30     public RiderManager riderManager = new RiderManager();
31
32     @Override
33     public int[] getRaceIds() {
34         return Race.getAllRaceIds();
35     }
36
37     @Override
38     public int createRace(String name, String description)
           throws IllegalNameException, InvalidNameException {
39         Race r = new Race(name, description);
40         return r.getRaceId();
41     }
42
```

```
43      @Override
44      public String viewRaceDetails(int raceId) throws
            IDNotRecognisedException {
45          double sum = 0.0;
46          for(int id : Race.getStages(raceId)) {
47              sum += Stage.getStageLength(id);
48          }
49          return Race.toString(raceId)+Double.toString(sum)+";";
50      }
51
52      @Override
53      public void removeRaceById(int raceId) throws
            IDNotRecognisedException {
54          Race.removeRace(raceId);
55      }
56
57      @Override
58      public int getNumberOfStages(int raceId) throws
            IDNotRecognisedException {
59          int[] stageIds = Race.getStages(raceId);
60          return stageIds.length;
61      }
62
63      @Override
64      public int addStageToRace(int raceId, String stageName,
            String description, double length, LocalDateTime
            startTime,
65              StageType type)
66              throws IDNotRecognisedException,
                    IllegalNameException, InvalidNameException,
                    InvalidLengthException {
67          return Race.addStageToRace(raceId, stageName,
                description, length, startTime, type);
68      }
69
70      @Override
71      public int[] getRaceStages(int raceId) throws
            IDNotRecognisedException {
72          return Race.getStages(raceId);
73      }
74
75      @Override
76      public double getStageLength(int stageId) throws
            IDNotRecognisedException {
77          return Stage.getStageLength(stageId);
78      }
79
80      @Override
81      public void removeStageById(int stageId) throws
            IDNotRecognisedException {
```

```
82          Race.removeStage(stageId);
83      }
84
85      @Override
86      public int addCategorizedClimbToStage(int stageId, Double
            location, SegmentType type, Double averageGradient,
87              Double length) throws IDNotRecognisedException,
                    InvalidLocationException,
                    InvalidStageStateException,
88              InvalidStageTypeException {
89          return Stage.addSegmentToStage(stageId, location, type,
                 averageGradient, length);
90      }
91
92      @Override
93      public int addIntermediateSprintToStage(int stageId, double
             location) throws IDNotRecognisedException,
94              InvalidLocationException,
                    InvalidStageStateException,
                    InvalidStageTypeException {
95          // TODO Check inputs?
96          return Stage.addSegmentToStage(stageId, location,
                SegmentType.SPRINT, 0.0, location);
97      }
98
99      @Override
100     public void removeSegment(int segmentId) throws
            IDNotRecognisedException, InvalidStageStateException {
101         Stage.removeSegment(segmentId);
102     }
103
104     @Override
105     public void concludeStagePreparation(int stageId) throws
            IDNotRecognisedException, InvalidStageStateException {
106         Stage.updateStageState(stageId);
107     }
108
109     @Override
110     public int[] getStageSegments(int stageId) throws
            IDNotRecognisedException {
111         return Stage.getSegments(stageId);
112     }
113
114     @Override
115     public int createTeam(String name, String description)
            throws IllegalNameException, InvalidNameException {
116         return riderManager.createTeam(name, description);
117     }
118
119     @Override
```

```
120     public void removeTeam(int teamId) throws
            IDNotRecognisedException {
121         riderManager.removeTeam(teamId);
122
123     }
124
125     @Override
126     public int[] getTeams() {
127         return riderManager.getTeams();
128     }
129
130     @Override
131     public int[] getTeamRiders(int teamId) throws
            IDNotRecognisedException {
132         return riderManager.getTeamRiders(teamId);
133     }
134
135     @Override
136     public int createRider(int teamID, String name, int
            yearOfBirth) throws IDNotRecognisedException,
            IllegalArgumentException {
137         return riderManager.createRider(teamID, name,
                yearOfBirth);
138
139     }
140
141     @Override
142     public void removeRider(int riderId) throws
            IDNotRecognisedException {
143         riderManager.removeRider(riderId);
144
145     }
146
147     @Override
148     public void registerRiderResultsInStage(int stageId, int
            riderId, LocalTime... checkpoints)
149             throws IDNotRecognisedException,
                    DuplicatedResultException,
                    InvalidCheckpointsException,
150                 InvalidStageStateException {
151         if(Stage.getStageState(stageId).equals(StageState.
                BUILDING)) {
152             throw new InvalidStageStateException("stage is not
                    waiting for results");
153         } else if(Stage.getSegments(stageId).length+2 !=
                checkpoints.length) {
154             throw new InvalidCheckpointsException("checkpoint
                    count mismatch");
155         }
156         try {
```

```
157             Result.getResult(stageId, riderId);
158             throw new DuplicatedResultException();
159         } catch(IDNotRecognisedException ex) {
160             Stage.getStage(stageId);
161             riderManager.getRider(riderId);
162             // above should throw exceptions if IDs are not in
                    system
163             new Result(stageId, riderId, checkpoints);
164         }
165     }
166
167     @Override
168     public LocalTime[] getRiderResultsInStage(int stageId, int
            riderId) throws IDNotRecognisedException {
169         Stage.getStage(stageId);
170         riderManager.getRider(riderId);
171         // above should throw exceptions if IDs are not in
                system
172         Result result = Result.getResult(stageId, riderId);
173         LocalTime[] checkpointTimes = result.getCheckpoints();
174         LocalTime[] out = new LocalTime[checkpointTimes.length
                +1];
175         for(int i=0; i<checkpointTimes.length; i++) {
176             out[i] = checkpointTimes[i];
177         }
178         out[-1] = result.getTotalElasped();
179         return out;
180     }
181
182     @Override
183     public LocalTime getRiderAdjustedElapsedTimeInStage(int
            stageId, int riderId) throws IDNotRecognisedException {
184         Stage.getStage(stageId);
185         riderManager.getRider(riderId);
186         // above should throw exceptions if IDs are not in
                system
187         LocalTime[] adjustedTimes = Result.getResult(stageId,
                riderId).adjustedCheckpoints();
188         LocalTime elapsedTime = adjustedTimes[0];
189         for(int i=1; i<adjustedTimes.length; i++) {
190             LocalTime t = adjustedTimes[i];
191             elapsedTime.plusHours(t.getHour()).plusMinutes(t.
                    getMinute()).plusSeconds(t.getSecond()).
                    plusNanos(t.getNano());
192         }
193         return elapsedTime;
194     }
195
196     @Override
197     public void deleteRiderResultsInStage(int stageId, int
```

```
            riderId) throws IDNotRecognisedException {
198         Stage.getStage(stageId);
199         riderManager.getRider(riderId);
200         // above should throw exceptions if IDs are not in
                system
201         Result.removeResult(stageId, riderId);
202     }
203
204     @Override
205     public int[] getRidersRankInStage(int stageId) throws
            IDNotRecognisedException {
206         Result[] results = Result.getResultsInStage(stageId);
207         int[] riderRanks = new int[results.length];
208         Arrays.fill(riderRanks, -1);
209         for(Result r : results) {
210             for(int i=0; i<riderRanks.length; i++) {
211                 if(riderRanks[i] == -1) {
212                     riderRanks[i] = r.getRiderId();
213                 } else {
214                     LocalTime[] rTimes = r.getCheckpoints();
215                     LocalTime[] compTimes = Result.getResult(
                            stageId, riderRanks[i]).getCheckpoints()
                            ;
216                     if(rTimes[rTimes.length-1].isBefore(
                            compTimes[compTimes.length-1])) {
217                         int temp;
218                         int prev = r.getRiderId();
219                         for(int j=i; j<riderRanks.length; j++)
                                {
220                             temp = riderRanks[j];
221                             riderRanks[j] = prev;
222                             prev = temp;
223                             if(prev == -1) {
224                                 break;
225                             }
226                         }
227                     }
228                     break;
229                 }
230             }
231         }
232         return riderRanks;
233     }
234
235     @Override
236     public LocalTime[] getRankedAdjustedElapsedTimesInStage(int
            stageId) throws IDNotRecognisedException {
237         // TODO Auto-generated method stub
238         // TODO Thomas do this after mountain points
239         return null;
```

```
240     }
241
242     @Override
243     public int[] getRidersPointsInStage(int stageId) throws
            IDNotRecognisedException {
244         StageType type = Stage.getStageType(stageId);
245         int[] points = new int[Result.getResultsInStage(stageId
                ).length];
246         int[] distribution = new int[15];
247         // distributions from https://en.wikipedia.org/wiki/
                Points_classification_in_the_Tour_de_France
248         switch(type) {
249             case FLAT:
250                 distribution = new int
                        []{50,30,20,18,16,14,12,10,8,7,6,5,4,3,2};
251                 break;
252             case MEDIUM_MOUNTAIN:
253                 distribution = new int
                        []{30,25,22,19,17,15,13,11,9,7,6,5,4,3,2};
254                 break;
255             case HIGH_MOUNTAIN:
256                 distribution = new int
                        []{20,17,15,13,11,10,9,8,7,6,5,4,3,2,1};
257                 break;
258             case TT:
259                 distribution = new int
                        []{20,17,15,13,11,10,9,8,7,6,5,4,3,2,1};
260                 break;
261         }
262         for(int i=0; i<Math.min(points.length, distribution.
                length); i++) {
263             points[i] = distribution[i];
264         }
265         return points;
266     }
267
268     @Override
269     public int[] getRidersMountainPointsInStage(int stageId)
            throws IDNotRecognisedException {
270         Result[] results = Result.getResultsInStage(stageId);
271         // All results refering to the stage with id *stageId*
272         int[] riders = getRidersRankInStage(stageId);
273         // An int array of rider ids, from first to last
274         int[] segments = Stage.getSegments(stageId);
275         // An int array of the segment ids in the stage
276         int[] points = new int[riders.length];
277         // The int in position i is the number of points to be
                awarded to the rider with id riders[i]
278         for(int s=0; s<segments.length; s++) {
279             SegmentType type = Segment.getSegmentType(segments[
```

```
                s]);
280             int[] distribution = new int[1];
281             // The points to be awarded in order for the
                    segment
282             switch(type) {
283                 case C4:
284                     distribution = new int[]{1};
285                     break;
286                 case C3:
287                     distribution = new int[]{2,1};
288                     break;
289                 case C2:
290                     distribution = new int[]{5,3,2,1};
291                     break;
292                 case C1:
293                     distribution = new int[]{10,8,6,4,2,1};
294                     break;
295                 case HC:
296                     distribution = new int
                            []{20,15,12,10,8,6,4,2};
297                     break;
298                 case SPRINT:
299             }
300             // get ranks for segment
301             int[] riderRanks = new int[results.length];
302             Arrays.fill(riderRanks, -1);
303             for(Result r : results) {
304                 for(int i=0; i<riderRanks.length; i++) {
305                     if(riderRanks[i] == -1) {
306                         riderRanks[i] = r.getRiderId();
307                     } else {
308                         Result compare = Result.getResult(
                                stageId, riderRanks[i]);
309                         if(r.getCheckpoints()[s].isBefore(
                                compare.getCheckpoints()[s])) {
310                             int temp;
311                             int prev = r.getRiderId();
312                             for(int j=i; j<riderRanks.length; j
                                    ++) {
313                                 temp = riderRanks[j];
314                                 riderRanks[j] = prev;
315                                 prev = temp;
316                                 if(prev == -1) {
317                                     break;
318                                 }
319                             }
320                         }
321                         break;
322                     }
323                 }
```

```
324             }
325             //return riderRanks;
326             ArrayList<Integer> ridersArray = new ArrayList<
                    Integer>();
327             for(int r : riders) { ridersArray.add(r); }
328             for(int i=0; i<Math.min(points.length, distribution
                    .length); i++) {
329                 int overallPos = ridersArray.indexOf(riderRanks
                        [i]);
330                 if(overallPos<points.length) {
331                     points[overallPos] += distribution[i];
332                 }
333             }
334         }
335         return points;
336     }
337
338     @Override
339     public void eraseCyclingPortal() {
340
341         Team.teamNames.clear();
342         Team.teamTopId = 0;
343         Rider.ridersTopId = 0;
344
345         RiderManager.allRiders.clear();
346         RiderManager.allTeams.clear();
347
348
349         Race.allRaces.clear();
350         Race.removedIds.clear();
351         Race.loadId();
352
353         Segment.allSegments.clear();
354         Segment.removedIds.clear();
355         Segment.loadId();
356
357         Stage.allStages.clear();
358         Stage.removedIds.clear();
359         Stage.loadId();
360
361         Result.allResults.clear();
362
363
364     }
365
366     @Override
367     public void saveCyclingPortal(String filename) throws
            IOException {
368         try {
369             FileOutputStream fos = new FileOutputStream(
```

```
                filename);
370             ObjectOutputStream oos = new ObjectOutputStream(fos
                    );
371             ArrayList<ArrayList> allObj = new ArrayList<>();
372             allObj.add(RiderManager.allTeams);
373             allObj.add(RiderManager.allRiders);
374             allObj.add(Stage.allStages);
375             allObj.add(Stage.removedIds);
376             allObj.add(Race.allRaces);
377             allObj.add(Race.removedIds);
378             allObj.add(Result.allResults);
379             allObj.add(Segment.allSegments);
380             allObj.add(Segment.removedIds);
381
382             oos.writeObject(allObj);
383
384             oos.flush();
385             oos.close();
386
387         } catch (IOException ex) {
388             ex.printStackTrace();
389         }
390
391     }
392
393     @Override
394     public void loadCyclingPortal(String filename) throws
            IOException, ClassNotFoundException {
395         try {
396
397             FileInputStream fis = new FileInputStream(filename)
                    ;
398             ObjectInputStream ois = new ObjectInputStream(fis);
399             ArrayList<Object> allObjects = new ArrayList<>();
400             ArrayList<Team> allTeams = new ArrayList<>();
401             ArrayList<Rider> allRiders = new ArrayList<>();
402             ArrayList<Result> allResults = new ArrayList<Result
                    >();
403             ArrayList<Race> allRaces = new ArrayList<Race>();
404             ArrayList<Stage> allStages = new ArrayList<Stage>()
                    ;
405             ArrayList<Segment> allSegments = new ArrayList<
                    Segment>();
406             ArrayList<Integer> removedIds = new ArrayList<>();
407
408             Class<?> classFlag = null;
409
410             allObjects = (ArrayList) ois.readObject();
411             for (Object tempObj : allObjects){
412                 ArrayList Objects = (ArrayList) tempObj;
```

```
413                 for (Object obj : Objects){
414                     if (classFlag != null){
415                         if (obj.getClass() != classFlag && obj.
                                getClass() != Integer.class){
416                             if (classFlag == Race.class){
417                                 Race.removedIds = removedIds;
418                             }
419                             if (classFlag == Segment.class){
420                                 Segment.removedIds = removedIds;
421                             }
422                             if (classFlag == Stage.class){
423                                 Stage.removedIds = removedIds;
424                             }
425                             classFlag = null;
426                             removedIds.clear();
427
428
429                         }
430                         else{
431                             Integer removedId = (Integer) obj;
432                             removedIds.add(removedId);
433
434                         }
435                     }
436                     String objClass = obj.getClass().getName();
437                     System.out.println(objClass);
438                     if (obj.getClass() == Rider.class){
439                         Rider newRider = (Rider) obj;
440                         allRiders.add(newRider);
441                         System.out.println("NEW RIDER");
442                     }
443                     if (obj.getClass() == Team.class){
444                         Team newTeam = (Team) obj;
445                         allTeams.add(newTeam);
446                         System.out.println("NEW TEAM");
447                     }
448                     if (obj.getClass() == Result.class){
449                         Result newResult = (Result) obj;
450                         allResults.add(newResult);
451                         System.out.println("NEW RESULT");
452                     }
453                     if (obj.getClass() == Stage.class){
454                         Stage newStage = (Stage) obj;
455                         allStages.add(newStage);
456                         System.out.println("NEW STAGE");
457                         classFlag = Stage.class;
458                     }
459                     if (obj.getClass() == Race.class){
460                         Race newRace = (Race) obj;
461                         allRaces.add(newRace);
```

```
462                     System.out.println("NEW Race");
463                     classFlag = Race.class;
464                 }
465                 if (obj.getClass() == Segment.class){
466                     Segment newSeg = (Segment) obj;
467                     allSegments.add(newSeg);
468                     System.out.println("NEW SEGMENT");
469                     classFlag = Segment.class;
470                 }
471
472
473                 System.out.println(obj.getClass());
474             }
475         }
476         if (classFlag == Race.class){
477             Race.removedIds = removedIds;
478         }
479         if (classFlag == Segment.class){
480             Segment.removedIds = removedIds;
481         }
482         if (classFlag == Stage.class){
483             Stage.removedIds = removedIds;
484         }
485
486             this.riderManager.setAllTeams(allTeams);
487             this.riderManager.setAllRiders(allRiders);
488             Race.allRaces = allRaces;
489             Race.loadId();
490             Stage.allStages = allStages;
491             Stage.loadId();
492             Segment.allSegments = allSegments;
493             Segment.loadId();
494             Result.allResults = allResults;
495             ois.close();
496
497         }
498         catch (Exception ex) {
499             ex.printStackTrace();
500         }
501
502     }
503
504     @Override
505     public void removeRaceByName(String name) throws
            NameNotRecognisedException {
506         boolean found = false;
507         for (int raceId : Race.getAllRaceIds()){ //Throwing
                this exception is impossible!
508             try {
509                 if (name == Race.getRaceName(raceId)){
```

```
510                     Race.removeRace(raceId);
511                 }
512             }
513             catch(Exception c){
514                 assert(false); //Assert false for this!
515             }
516 
517         }
518         if (!found){ throw new NameNotRecognisedException("Name
                not in System.");}
519 
520     }
521 
522     @Override
523     public LocalTime[] getGeneralClassificationTimesInRace(int
            raceId) throws IDNotRecognisedException {
524         // TODO Auto-generated method stub
525         return null;
526     }
527 
528     @Override
529     public int[] getRidersPointsInRace(int raceId) throws
            IDNotRecognisedException {
530         // TODO Auto-generated method stub
531         return null;
532     }
533 
534     @Override
535     public int[] getRidersMountainPointsInRace(int raceId)
            throws IDNotRecognisedException {
536         // TODO Auto-generated method stub
537         return null;
538     }
539 
540     @Override
541     public int[] getRidersGeneralClassificationRank(int raceId)
             throws IDNotRecognisedException {
542         // TODO Auto-generated method stub
543         return null;
544     }
545 
546     @Override
547     public int[] getRidersPointClassificationRank(int raceId)
            throws IDNotRecognisedException {
548         // TODO Auto-generated method stub
549         return null;
550     }
551 
552     @Override
553     public int[] getRidersMountainPointClassificationRank(int
```

```
            raceId) throws IDNotRecognisedException {
554         // TODO Auto-generated method stub
555         return null;
556     }
557
558 }
```

2 Race.java

```
1  package cycling;
2
3  import java.util.ArrayList;
4  import java.io.Serializable;
5  import java.time.LocalDateTime;
6
7  /**
8   * Race encapsulates tour races, each of which has a number of
        associated
9   * Stages.
10  *
11  * @author Thomas Newbold
12  * @version 2.0
13  *
14  */
15 public class Race implements Serializable {
16     // Static class attributes
17     private static int idMax = 0;
18     public static ArrayList<Integer> removedIds = new ArrayList
           <Integer>();
19     public static ArrayList<Race> allRaces = new ArrayList<Race
           >();
20
21     /**
22      * Loads the value of idMax.
23      */
24     public static void loadId(){
25         if(Race.allRaces.size()!=0) {
26             Race.idMax = Race.allRaces.get(Race.allRaces.size()
                   -1).getRaceId() + 1;
27         } else {
28             Race.idMax = 0;
29         }
30     }
31
32     /**
33      * @param raceId The ID of the race instance to fetch
34      * @return The race instance with the associated ID
35      * @throws IDNotRecognisedException If no race exists with
            the requested ID
```

```
36          */
37      public static Race getRace(int raceId) throws
            IDNotRecognisedException {
38          boolean removed = Race.removedIds.contains(raceId);
39          if(raceId<Race.idMax && raceId >= 0 && !removed) {
40              int index = raceId;
41              for(int j=0; j<Race.removedIds.size(); j++) {
42                  if(Race.removedIds.get(j) < raceId) {
43                      index--;
44                  }
45              }
46              return allRaces.get(index);
47          } else if (removed) {
48              throw new IDNotRecognisedException("no race
                    instance for raceID");
49          } else {
50              throw new IDNotRecognisedException("raceID out of
                    range");
51          }
52      }
53
54      /**
55       * @return An integer array of the race IDs of all races
56       */
57      public static int[] getAllRaceIds() {
58          int length = Race.allRaces.size();
59          int[] raceIdsArray = new int[length];
60          int i = 0;
61          for(Race race : allRaces) {
62              raceIdsArray[i] = race.getRaceId();
63              i++;
64          }
65          return raceIdsArray;
66      }
67
68      /**
69       * @param raceId The ID of the race instance to remove
70       * @throws IDNotRecognisedException If no race exists with
               the requested ID
71       */
72      public static void removeRace(int raceId) throws
            IDNotRecognisedException {
73          boolean removed = Race.removedIds.contains(raceId);
74          if(raceId<Race.idMax && raceId >= 0 && !removed) {
75              Race r = getRace(raceId);
76              for(int id : r.getStages()) {
77                  r.removeStageFromRace(id);
78              }
79              allRaces.remove(raceId);
80              removedIds.add(raceId);
```

```
81          } else if (removed) {
82              throw new IDNotRecognisedException("no race
                    instance for raceID");
83          } else {
84              throw new IDNotRecognisedException("raceID out of
                    range");
85          }
86      }
87
88      // Instance attributes
89      private int raceId;
90      private String raceName;
91      private String raceDescription;
92      private ArrayList<Integer> stageIds;
93
94      /**
95       * @param name String to be checked
96       * @return true if name is valid for the system
97       */
98      private static boolean validName(String name) {
99          if(name==null || name.equals("")) {
100             return false;
101         } else if(name.length()>30) {
102             return false;
103         } else if(name.contains(" ")) {
104             return false;
105         } else {
106             return true;
107         }
108     }
109
110     /**
111      * Race constructor; creates new race and adds to allRaces
             array.
112      *
113      * @param name The name of the new race
114      * @param description The description for the new race
115      * @throws IllegalNameException If name already exists in
             the system
116      * @throws InvalidNameException If name is empty/null,
             contains whitespace,
117      *                              or is longer than 30
             characters
118      */
119     public Race(String name, String description) throws
            IllegalNameException,
120                     InvalidNameException {
121         for(Race race : allRaces) {
122             if(race.getRaceName().equals(name)) {
123                 throw new IllegalNameException("name already
```

```
                    exists");
124             }
125         }
126         if(!validName(name)) {
127             throw new InvalidNameException("invalid name");
128         }
129         if(Race.removedIds.size() > 0) {
130             this.raceId = Race.removedIds.get(0);
131             Race.removedIds.remove(0);
132         } else {
133             this.raceId = idMax++;
134         }
135         this.raceName = name;
136         this.raceDescription = description;
137         this.stageIds = new ArrayList<Integer>();
138         Race.allRaces.add(this);
139     }
140 
141     /**
142      * @return A string representation of the race instance
143      */
144     public String toString() {
145         String id = Integer.toString(this.raceId);
146         String name = this.raceName;
147         String description = this.raceDescription;
148         String list = this.stageIds.toString();
149         return String.format("Race[%s]: %s; %s; StageIds=%s;",
                id, name,
150                              description, list);
151     }
152 
153     /**
154      * @param id The ID of the race
155      * @return A string representation of the race instance
156      * @throws IDNotRecognisedException If no race exists with
            the requested ID
157      */
158     public static String toString(int id) throws
            IDNotRecognisedException {
159         return getRace(id).toString();
160     }
161 
162     /**
163      * @return The integer raceId for the race instance
164      */
165     public int getRaceId() { return this.raceId; }
166 
167     /**
168      * @return The string raceName for the race instance
169      */
```

```
170     public String getRaceName() { return this.raceName; }
171
172     /**
173      * @param id The ID of the race
174      * @return The string raceName for the race with the
             associated id
175      * @throws IDNotRecognisedException If no race exists with
             the requested ID
176      */
177     public static String getRaceName(int id) throws
            IDNotRecognisedException {
178         return getRace(id).raceName;
179     }
180
181     /**
182      * @return The string raceDescription for the race instance
183      */
184     public String getRaceDescription() { return this.
            raceDescription; }
185
186     /**
187      * @param id The ID of the race
188      * @return The string raceDescription for the race with the
             associated id
189      * @throws IDNotRecognisedException If no race exists with
             the requested ID
190      */
191     public static String getRaceDescription(int id) throws
192                                                 IDNotRecognisedException
                                                        {
193         return getRace(id).raceDescription;
194     }
195
196     /**
197      * @return An integer array of stage IDs for the race
             instance
198      */
199     public int[] getStages() {
200         int length = this.stageIds.size();
201         int[] stageIdsArray = new int[length];
202         for(int i=0; i<length; i++) {
203             stageIdsArray[i] = this.stageIds.get(i);
204         }
205         return stageIdsArray;
206     }
207
208     /**
209      * @param id The ID of the race
210      * @return An integer array of stage IDs for the race
             instance
```

```
211         * @throws IDNotRecognisedException If no race exists with
                the requested ID
212         */
213        public static int[] getStages(int id) throws
               IDNotRecognisedException {
214            Race race = getRace(id);
215            int length = race.stageIds.size();
216            int[] stageIdsArray = new int[length];
217            for(int i=0; i<length; i++) {
218                stageIdsArray[i] = race.stageIds.get(i);
219            }
220            return stageIdsArray;
221        }
222
223        /**
224         * @param name The new name for the race instance
225         */
226        public void setRaceName(String name) {
227            this.raceName = name;
228        }
229
230        /**
231         * @param id The ID of the race to be updated
232         * @param name The new name for the race instance
233         * @throws IDNotRecognisedException If no race exists with
                the requested ID
234         */
235        public static void setRaceName(int id, String name) throws
236                                           IDNotRecognisedException {
237            getRace(id).setRaceName(name);
238        }
239
240        /**
241         * @param description The new description for the race
                instance
242         */
243        public void setRaceDescription(String description) {
244            this.raceDescription = description;
245        }
246
247        /**
248         * @param id The ID of the race to be updated
249         * @param description The new description for the race
                instance
250         * @throws IDNotRecognisedException If no race exists with
                the requested ID
251         */
252        public static void setRaceDescription(int id, String
               description) throws
253                                                IDNotRecognisedException
```

```
                                                    {
254             getRace(id).setRaceDescription(description);
255         }
256
257         /**
258          * Creates a new stage and adds the ID to the stageIds
                array.
259          *
260          * @param name The name of the new stage
261          * @param description The description of the new stage
262          * @param length The length of the new stage (in km)
263          * @param startTime The date and time at which the stage
                will be held
264          * @param type The StageType, used to determine the point
                distribution
265          * @return The ID of the new stage
266          */
267         public int addStageToRace(String name, String description,
                double length,
268                                   LocalDateTime startTime,
                                        StageType type) throws
269                                   IllegalNameException,
                                        InvalidNameException,
270                                   InvalidLengthException {
271             Stage newStage = new Stage(name, description, length,
                    startTime, type);
272             this.stageIds.add(newStage.getStageId());
273             return newStage.getStageId();
274         }
275
276         /**
277          * Creates a new stage and adds the ID to the stageIds
                array.
278          *
279          * @param id The ID of the race to which the stage will be
                added
280          * @param name The name of the new stage
281          * @param description The description of the new stage
282          * @param length The length of the new stage (in km)
283          * @param startTime The date and time at which the stage
                will be held
284          * @param type The StageType, used to determine the point
                distribution
285          * @return The ID of the new stage
286          * @throws IDNotRecognisedException If no race exists with
                the requested ID
287          */
288         public static int addStageToRace(int id, String name,
                String description,
289                                          double length,
```

```
                                                LocalDateTime startTime
                                                ,
290                                             StageType type) throws
291                                             IDNotRecognisedException,
292                                             IllegalNameException,
                                                    InvalidNameException,
293                                             InvalidLengthException  {
294         return getRace(id).addStageToRace(name, description,
                length, startTime, type);
295     }
296
297     /**
298      * Removes a stageId from the array of stageIds for a race
             instance,
299      * as well as from the static array of all stages in the
             Stage class.
300      *
301      * @param stageId The ID of the stage to be removed
302      * @throws IDNotRecognisedException If no stage exists with
             the requested ID
303      */
304     private void removeStageFromRace(int stageId) throws
            IDNotRecognisedException {
305         if(this.stageIds.contains(stageId)) {
306             this.stageIds.remove(stageId);
307             Stage.removeStage(stageId);
308         } else {
309             throw new IDNotRecognisedException("stageID not
                    found in race");
310         }
311     }
312
313     /**
314      * Removes a stageId from the array of stageIds for a race
             instance,
315      * as well as from the static array of all stages in the
             Stage class.
316      *
317      * @param id The ID of the race to which the stage will be
             removed
318      * @param stageId The ID of the stage to be removed
319      * @throws IDNotRecognisedException If no stage exists with
             the requested ID
320      */
321     public static void removeStageFromRace(int id, int stageId)
             throws
322                                                 IDNotRecognisedException
                                                        {
323         getRace(id).removeStageFromRace(stageId);
324     }
```

```
325
326     /**
327      * Removes a stageId from the array of stageIds for a race
            instance,
328      * as well as from the static array of all stages in the
            Stage class.
329      *
330      * @param stageId The ID of the stage to be removed
331      * @throws IDNotRecognisedException If no stage exists with
            the requested ID
332      */
333     public static void removeStage(int stageId) throws
            IDNotRecognisedException {
334         for(Race race : allRaces) {
335             if(race.stageIds.contains(stageId)) {
336                 race.removeStageFromRace(stageId);
337                 break;
338             }
339         }
340     }
341 }
```

3 Stage.java

```
1  package cycling;
2
3  import java.util.ArrayList;
4  import java.io.Serializable;
5  import java.time.LocalDateTime;
6  import java.time.format.DateTimeFormatter;
7
8  /**
9   * Stage encapsulates race stages, each of which has a number
        of associated
10  * Segments.
11  *
12  * @author Thomas Newbold
13  * @version 2.0
14  *
15  */
16 public class Stage implements Serializable {
17     // Static class attributes
18     private static int idMax = 0;
19     public static ArrayList<Integer> removedIds = new ArrayList
           <Integer>();
20     public static ArrayList<Stage> allStages = new ArrayList<
           Stage>();
21
22     /**
```

```
23          * Loads the value of idMax.
24          */
25         public static void loadId(){
26             if(Stage.allStages.size()!=0) {
27                 Stage.idMax = Stage.allStages.get(Stage.allStages.
                       size()-1).getStageId() + 1;
28             } else {
29                 Stage.idMax = 0;
30             }
31         }
32
33         /**
34          * @param stageId The ID of the stage instance to fetch
35          * @return The stage instance with the associated ID
36          * @throws IDNotRecognisedException If no stage exists with
                the requested ID
37          */
38         public static Stage getStage(int stageId) throws
               IDNotRecognisedException {
39             boolean removed = Stage.removedIds.contains(stageId);
40             if(stageId<Stage.idMax && stageId >= 0 && !removed) {
41                 int index = stageId;
42                 for(int j=0; j<Stage.removedIds.size(); j++) {
43                     if(Stage.removedIds.get(j) < stageId) {
44                         index--;
45                     }
46                 }
47                 return allStages.get(index);
48             } else if (removed) {
49                 throw new IDNotRecognisedException("no stage
                       instance for stageID");
50             } else {
51                 throw new IDNotRecognisedException("stageId out of
                       range");
52             }
53         }
54
55         /**
56          * @return An integer array of the stage IDs of all stage
57          */
58         public static int[] getAllStageIds() {
59             int length = Stage.allStages.size();
60             int[] stageIdsArray = new int[length];
61             int i = 0;
62             for(Stage stage : allStages) {
63                 stageIdsArray[i] = stage.getStageId();
64                 i++;
65             }
66             return stageIdsArray;
67         }
```

```
68
69      /**
70       * @param stageId The ID of the stage instance to remove
71       * @throws IDNotRecognisedException If no stage exists with
             the requested ID
72       */
73      public static void removeStage(int stageId) throws
            IDNotRecognisedException {
74          boolean removed = Stage.removedIds.contains(stageId);
75          if(stageId<Stage.idMax && stageId >= 0 && !removed) {
76              Stage s = getStage(stageId);
77              for(int id : s.getSegments()) {
78                  s.removeSegmentFromStage(id);
79              }
80              allStages.remove(stageId);
81              removedIds.add(stageId);
82          } else if (removed) {
83              throw new IDNotRecognisedException("no stage
                    instance for stageID");
84          } else {
85              throw new IDNotRecognisedException("stageId out of
                    range");
86          }
87      }
88
89      // Instance attributes
90      private int stageId;
91      private StageState stageState;
92      private String stageName;
93      private String stageDescription;
94      private double stageLength;
95      private LocalDateTime stageStartTime;
96      private StageType stageType;
97      private ArrayList<Integer> segmentIds;
98
99      /**
100      * @param name String to be checked
101      * @return true if name is valid for the system
102      */
103     private static boolean validName(String name) {
104         if(name==null || name.equals("")) {
105             return false;
106         } else if(name.length()>30) {
107             return false;
108         } else if(name.contains(" ")) {
109             return false;
110         } else {
111             return true;
112         }
113     }
```

```
114
115         /**
116          * Stage constructor; creates a new stage and adds to
                allStages array.
117          *
118          * @param name The name of the new stage
119          * @param description The description of the new stage
120          * @param length The total length of the new stage
121          * @param startTime The start time for the new stage
122          * @param type The type of the new stage
123          * @throws IllegalNameException If name already exists in
                the system
124          * @throws InvalidNameException If name is empty/null,
                contains whitespace,
125          *                              or is longer than 30
                characters
126          * @throws InvalidLengthException If the length is less
                than 5km
127          */
128         public Stage(String name, String description, double length
                ,
129                      LocalDateTime startTime, StageType type)
                           throws
130                      IllegalNameException, InvalidNameException,
131                      InvalidLengthException {
132             for(Stage stage : allStages) {
133                 if(stage.getStageName().equals(name)) {
134                     throw new IllegalNameException("name already
                            exists");
135                 }
136             }
137             if(!validName(name)) {
138                 throw new InvalidNameException("invalid name");
139             }
140             if(length<5) {
141                 throw new InvalidLengthException("length less than
                        5km");
142             }
143             if(Stage.removedIds.size() > 0) {
144                 this.stageId = Stage.removedIds.get(0);
145                 Stage.removedIds.remove(0);
146             } else {
147                 this.stageId = idMax++;
148             }
149             this.stageState = StageState.BUILDING;
150             this.stageName = name;
151             this.stageDescription = description;
152             this.stageLength = length;
153             this.stageStartTime = startTime;
154             this.stageType = type;
```

```
155             this.segmentIds = new ArrayList<Integer>();
156             Stage.allStages.add(this);
157         }
158 
159         /**
160          * @return A string representation of the stage instance
161          */
162         public String toString() {
163             String id = Integer.toString(this.stageId);
164             String state;
165             switch (this.stageState) {
166                 case BUILDING:
167                     state = "In preperation";
168                     break;
169                 case WAITING:
170                     state = "Waiting for results";
171                     break;
172                 default:
173                     state = "null state";
174             }
175             String name = this.stageName;
176             String description = this.stageDescription;
177             String length = Double.toString(this.stageLength);
178             DateTimeFormatter formatter = DateTimeFormatter.
                    ofPattern("HH:hh dd-MM-yyyy");
179             String startTime = this.stageStartTime.format(formatter
                    );
180             String list = this.segmentIds.toString();
181             String type;
182             switch (this.stageType) {
183                 case FLAT:
184                     type = "Flat";
185                     break;
186                 case MEDIUM_MOUNTAIN:
187                     type = "Medium Mountain";
188                     break;
189                 case HIGH_MOUNTAIN:
190                     type = "High Mountain";
191                     break;
192                 case TT:
193                     type = "Time Trial";
194                     break;
195                 default:
196                     type = "null type";
197             }
198             return String.format("Stage[%s](%s): %s (%s); %s; %skm;
                     %s; SegmentIds=%s;",
199                                  id, state, name, type, description
                                      , length,
200                                  startTime, list);
```

```
201     }
202
203     /**
204      * @param id The ID of the stage
205      * @return A string representation of the stage instance
206      * @throws IDNotRecognisedException If no stage exists with
             the requested ID
207      */
208     public static String toString(int id) throws
            IDNotRecognisedException {
209         return getStage(id).toString();
210     }
211
212     /**
213      * @return The integer stageId for the stage instance
214      */
215     public int getStageId() { return this.stageId; }
216
217     /**
218      * @return The state of the stage instance
219      */
220     public StageState getStageState() { return this.stageState;
            }
221
222     /**
223      * @param id The ID of the stage
224      * @return The state of the stage instance
225      * @throws IDNotRecognisedException If no stage exists with
             the requested ID
226      */
227     public static StageState getStageState(int id) throws
228                                                 IDNotRecognisedException
                                                        {
229         return getStage(id).getStageState();
230     }
231     /**
232      * @return The string raceName for the stage instance
233      */
234     public String getStageName() { return this.stageName; }
235
236     /**
237      * @param id The ID of the stage
238      * @return The string stageName for the stage with the
            associated id
239      * @throws IDNotRecognisedException If no stage exists with
             the requested ID
240      */
241     public static String getStageName(int id) throws
            IDNotRecognisedException {
242         return getStage(id).stageName;
```

```
243         }
244
245         /**
246          * @return The string stageDescription for the stage
                instance
247          */
248         public String getStageDescription() { return this.
                stageDescription; }
249
250         /**
251          * @param id The ID of the stage
252          * @return The string stageDescription for the stage with
                the associated id
253          * @throws IDNotRecognisedException If no stage exists with
                the requested ID
254          */
255         public static String getStageDescription(int id) throws
256                                                     IDNotRecognisedException
                                                         {
257             return getStage(id).stageDescription;
258         }
259
260         /**
261          * @return The length of the stage instance
262          */
263         public double getStageLength() { return this.stageLength; }
264
265         /**
266          * @param id The ID of the stage
267          * @return The length of the stage instance
268          * @throws IDNotRecognisedException If no stage exists with
                the requested ID
269          */
270         public static double getStageLength(int id) throws
                IDNotRecognisedException {
271             return getStage(id).stageLength;
272         }
273
274         /**
275          * @return The start time for the stage instance
276          */
277         public LocalDateTime getStageStartTime() { return this.
                stageStartTime; }
278
279         /**
280          * @param id The ID of the stage
281          * @return The start time for the stage instance
282          * @throws IDNotRecognisedException If no stage exists with
                the requested ID
283          */
```

```
284     public static LocalDateTime getStageStartTime(int id)
            throws
285                                                     IDNotRecognisedException
                                                            {
286         return getStage(id).stageStartTime;
287     }
288
289     /**
290      * @return The type of the stage instance
291      */
292     public StageType getStageType() { return this.stageType; }
293
294     /**
295      * @param id The ID of the stage
296      * @return The type of the stage instance
297      * @throws IDNotRecognisedException If no stage exists with
                 the requested ID
298      */
299     public static StageType getStageType(int id) throws
            IDNotRecognisedException {
300         return getStage(id).getStageType();
301     }
302
303     /**
304      * @return An integer array of segment IDs for the stage
             instance
305      */
306     public int[] getSegments() {
307         int length = this.segmentIds.size();
308         int[] segmentIdsArray = new int[length];
309         for(int i=0; i<length; i++) {
310             segmentIdsArray[i] = this.segmentIds.get(i);
311         }
312         return segmentIdsArray;
313     }
314
315     /**
316      * @param id The ID of the stage
317      * @return An integer array of segment IDs for the stage
             instance
318      * @throws IDNotRecognisedException If no stage exists with
                 the requested ID
319      */
320     public static int[] getSegments(int id) throws
            IDNotRecognisedException {
321         Stage stage = getStage(id);
322         int length = stage.segmentIds.size();
323         int[] segmentIdsArray = new int[length];
324         for(int i=0; i<length; i++) {
325             segmentIdsArray[i] = stage.segmentIds.get(i);
```

```
326             }
327             return segmentIdsArray;
328         }
329 
330         /**
331          * Updates the stage state from building to waiting for
                results.
332          *
333          * @throws InvalidStageStateException If the stage is
                already waiting for results
334          */
335         public void updateStageState() throws
                InvalidStageStateException {
336             if(this.stageState.equals(StageState.WAITING)) {
337                 throw new InvalidStageStateException("stage is
                        already waiting for results");
338             } else if(this.stageState.equals(StageState.BUILDING))
                    {
339                 this.stageState = StageState.WAITING;
340             }
341         }
342 
343         /**
344          * Updates the stage state from building to waiting for
                results.
345          *
346          * @param id The ID of the stage to be updated
347          * @throws IDNotRecognisedException If no stage exists with
                 the requested ID
348          * @throws InvalidStageStateException If the stage is
                already waiting for results
349          */
350         public static void updateStageState(int id) throws
                IDNotRecognisedException,
351                                             InvalidStageStateException
                                                    {
352             getStage(id).updateStageState();
353         }
354 
355         /**
356          * @param name The new name for the stage instance
357          */
358         public void setStageName(String name) {
359             this.stageName = name;
360         }
361 
362         /**
363          * @param id The ID of the stage to be updated
364          * @param name The new name for the stage instance
365          * @throws IDNotRecognisedException If no stage exists with
```

```
366             the requested ID
367          */
368         public static void setStageName(int id, String name) throws
369                                             IDNotRecognisedException {
370             getStage(id).setStageName(name);
371         }
372
373         /**
374          * @param description The new description for the stage
                instance
375          */
376         public void setStageDescription(String description) {
377             this.stageDescription = description;
378         }
379
380         /**
381          * @param id The ID of the stage to be updated
382          * @param description The new description for the stage
                instance
383          * @throws IDNotRecognisedException If no stage exists with
                the requested ID
384          */
385         public static void setStageDescription(int id, String
                description) throws
386                                                 IDNotRecognisedException
                                                        {
387             getStage(id).setStageDescription(description);
388         }
389
390         /**
391          * @param length The new length for the stage instance
392          */
393         public void setStageLength(double length) {
394             this.stageLength = length;
395         }
396
397         /**
398          * @param id The ID of the stage to be updated
399          * @param length The new length for the stage instance
400          * @throws IDNotRecognisedException If no stage exists with
                the requested ID
401          */
402         public static void setStageLength(int id, double length)
                throws
403                                             IDNotRecognisedException
                                                {
404             getStage(id).stageLength = length;
405         }
406
407         /**
```

```
407         * @param startTime The new start time for the stage
                instance
408         */
409        public void setStageStartTime(LocalDateTime startTime) {
410            this.stageStartTime = startTime;
411        }
412
413        /**
414         * @param id The ID of the stage to be updated
415         * @param startTime The new start time for the stage
                instance
416         * @throws IDNotRecognisedException If no stage exists with
                the requested ID
417         */
418        public static void setStageStartTime(int id, LocalDateTime
               startTime)
419                                             throws
                                                    IDNotRecognisedException
                                                     {
420            getStage(id).stageStartTime = startTime;
421        }
422
423        /**
424         * Creates a new stage and adds the ID to the stageIds
                array.
425         *
426         * @param location The location of the new segment
427         * @param type The type of the new segment
428         * @param averageGradient The average gradient of the new
                segment
429         * @param length The length (in km) of the new segment
430         * @throws InvalidLocationException If the segment finishes
                outside of the
431         *                                  bounds of the stage
432         * @throws InvalidStageStateException If the segment state
                is waiting for
433         *                                   results
434         * @throws InvalidStageTypeException If the stage type is a
                time-trial
435         *                                   (cannot contain
                segments)
436         */
437        public int addSegmentToStage(double location, SegmentType
               type,
438                                     double averageGradient, double
                                            length) throws
439                                     InvalidLocationException,
440                                     InvalidStageStateException,
441                                     InvalidStageTypeException {
442            if(location > this.getStageLength()) {
```

```
443             throw new InvalidLocationException("segment
                    finishes outside of stage bounds");
444         }
445         if(this.getStageState().equals(StageState.WAITING)) {
446             throw new InvalidStageStateException("stage is
                    waiting for results");
447         }
448         if(this.getStageType().equals(StageType.TT)) {
449             throw new InvalidStageTypeException("time trial
                    stages cannot contain segments");
450         }
451         Segment newSegment = new Segment(location, type,
                averageGradient, length);
452         this.segmentIds.add(newSegment.getSegmentId());
453         return newSegment.getSegmentId();
454     }
455
456     /**
457      * Creates a new stage and adds the ID to the stageIds
            array.
458      *
459      * @param id The ID of the stage to which the segment will
            be added
460      * @param location The location of the new segment
461      * @param type The type of the new segment
462      * @param averageGradient The average gradient of the new
            segment
463      * @param length The length (in km) of the new segment
464      * @throws IDNotRecognisedException If no stage exists with
            the requested ID
465      * @throws InvalidLocationException If the segment finishes
            outside of the
466      *                                  bounds of the stage
467      * @throws InvalidStageStateException If the segment state
            is waiting for
468      *                                   results
469      * @throws InvalidStageTypeException If the stage type is a
            time-trial
470      *                                   (cannot contain
            segments)
471      */
472     public static int addSegmentToStage(int id, double location
            , SegmentType type,
473                                         double averageGradient,
                                                double length)
                                            throws
474                                         IDNotRecognisedException
                                                ,
475                                         InvalidLocationException
                                                ,
```

```
476                                         InvalidStageStateException
                                                ,
477                                         InvalidStageTypeException
                                                {
478         return getStage(id).addSegmentToStage(location, type,
                averageGradient, length);
479     }
480
481     /**
482      * Removes a segmentId from the array of segmentIds for a
             stage instance,
483      * as well as from the static array of all segments in the
             Segment class.
484      *
485      * @param segmentId The ID of the segment to be removed
486      * @throws IDNotRecognisedException If no segment exists
             with the requested
487      *                                  ID
488      */
489     private void removeSegmentFromStage(int segmentId) throws
490                                         IDNotRecognisedException
                                                {
491         if(this.segmentIds.contains(segmentId)) {
492             this.segmentIds.remove(segmentId);
493             Segment.removeSegment(segmentId);
494         } else {
495             throw new IDNotRecognisedException("segmentID not
                    found in race");
496         }
497     }
498
499     /**
500      * Removes a segmentId from the array of segmentIds for a
             stage instance,
501      * as well as from the static array of all segments in the
             Segment class.
502      *
503      * @param id The ID of the stage to which the segment will
             be removed
504      * @param segmentId The ID of the segment to be removed
505      * @throws IDNotRecognisedException If no segment exists
             with the requested
506      *                                  ID
507      */
508     public static void removeSegmentFromStage(int id, int
            segmentId) throws
509                                             IDNotRecognisedException
                                                    {
510         getStage(id).removeSegmentFromStage(segmentId);
511     }
```

```
512
513     /**
514      * Removes a segmentId from the array of segmentIds for a
             stage instance,
515      * as well as from the static array of all segments in the
             Segment class.
516      *
517      * @param segmentId The ID of the segment to be removed
518      * @throws IDNotRecognisedException If no segment exists
             with the requested
519      *                                  ID
520      */
521     public static void removeSegment(int segmentId) throws
            IDNotRecognisedException {
522         for(Stage stage : allStages) {
523             if(stage.segmentIds.contains(segmentId)) {
524                 stage.removeSegmentFromStage(segmentId);
525                 break;
526             }
527         }
528     }
529 }
```

4 StageState.java

```
1  package cycling;
2
3  /**
4   * This enum is used to represent the state of a stage.
5   *
6   * @author Thomas Newbold
7   * @version 1.0
8   *
9   */
10 public enum StageState {
11
12     /**
13      * Used for stages still in preperation - i.e. segments are
             still being
14      * added.
15      */
16     BUILDING,
17
18     /**
19      * Used for stages waiting for results
20      */
21     WAITING;
22 }
```

5 Segment.java

```
1  package cycling;
2
3  import java.io.Serializable;
4  import java.util.ArrayList;
5
6  /**
7   * Segment encapsulates race segments
8   *
9   * @author Thomas Newbold
10  * @version 2.0
11  *
12  */
13 public class Segment implements Serializable {
14     // Static class attributes
15     private static int idMax = 0;
16     public static ArrayList<Integer> removedIds = new ArrayList
           <Integer>();
17     public static ArrayList<Segment> allSegments = new
           ArrayList<Segment>();
18
19     /**
20      * Loads the value of idMax.
21      */
22     public static void loadId(){
23         if(Segment.allSegments.size()!=0) {
24             Segment.idMax = Segment.allSegments.get(-1).
                   getSegmentId() + 1;
25         } else {
26             Segment.idMax = 0;
27         }
28     }
29
30     /**
31      * @param segmentId The ID of the segment instance to fetch
32      * @return The segment instance with the associated ID
33      * @throws IDNotRecognisedException If no segment exists
            with the requested
34      *                                      ID
35      */
36     public static Segment getSegment(int segmentId) throws
37                                         IDNotRecognisedException {
38         boolean removed = Segment.removedIds.contains(segmentId
               );
39         if(segmentId<Segment.idMax && segmentId >= 0 && !
               removed) {
40             int index = segmentId;
41             for(int j=0; j<Segment.removedIds.size(); j++) {
```

```
42                  if(Segment.removedIds.get(j) < segmentId) {
43                      index--;
44                  }
45              }
46              return allSegments.get(index);
47          } else if (removed) {
48              throw new IDNotRecognisedException("no segment
                    instance for "+
49                                                 "segmentId");
50          } else {
51              throw new IDNotRecognisedException("segmentId out
                    of range");
52          }
53      }
54
55      /**
56       * @return An integer array of the segment IDs of all
             segment
57       */
58      public static int[] getAllSegmentIds() {
59          int length = Segment.allSegments.size();
60          int[] segmentIdsArray = new int[length];
61          int i = 0;
62          for(Segment segment : allSegments) {
63              segmentIdsArray[i] = segment.getSegmentId();
64              i++;
65          }
66          return segmentIdsArray;
67      }
68
69      /**
70       * @param segmentId The ID of the segment instance to
             remove
71       * @throws IDNotRecognisedException If no segment exists
             with the requested
72       *                                  ID
73       */
74      public static void removeSegment(int segmentId) throws
75                                       IDNotRecognisedException {
76          boolean removed = Segment.removedIds.contains(segmentId
                );
77          if(segmentId<Segment.idMax && segmentId >= 0 && !
                removed) {
78              allSegments.remove(segmentId);
79              Segment.idMax--;
80              for(int i=segmentId;i<allSegments.size();i++) {
81                  getSegment(i).segmentId--;
82              }
83          } else if (removed) {
84              throw new IDNotRecognisedException("no segment
```

```
                instance for "+
85                                                      "segmentId");
86          } else {
87              throw new IDNotRecognisedException("segmentId out
                    of range");
88          }
89      }
90
91      // Instance attributes
92      private int segmentId;
93      private double segmentLocation;
94      private SegmentType segmentType;
95      private double segmentAverageGradient;
96      private double segmentLength;
97
98      /**
99       * Segment constructor; creates a new segment and adds to
             allSegment array.
100      *
101      * @param location The location of the finish of the new
             segment in the stage
102      * @param type The type of the new segment
103      * @param averageGradient The average gradient of the new
             segment
104      * @param length The length of the new segment
105      */
106     public Segment(double location, SegmentType type, double
            averageGradient,
107                    double length) {
108         if(Segment.removedIds.size() > 0) {
109             this.segmentId = Segment.removedIds.get(0);
110             Segment.removedIds.remove(0);
111         } else {
112             this.segmentId = idMax++;
113         }
114         this.segmentLocation = location;
115         this.segmentType = type;
116         this.segmentAverageGradient = averageGradient;
117         this.segmentLength = length;
118         Segment.allSegments.add(this);
119     }
120
121     /**
122      * @return A string representation of the segment instance
123      */
124     public String toString() {
125         String id = Integer.toString(this.segmentId);
126         String location = Double.toString(this.segmentLocation)
                ;
127         String type;
```

```
128             switch (this.segmentType) {
129                 case SPRINT:
130                     type = "Sprint";
131                     break;
132                 case C4:
133                     type = "Category 4 Climb";
134                     break;
135                 case C3:
136                     type = "Category 3 Climb";
137                     break;
138                 case C2:
139                     type = "Category 2 Climb";
140                     break;
141                 case C1:
142                     type = "Category 1 Climb";
143                     break;
144                 case HC:
145                     type = "Hors Categorie";
146                     break;
147                 default:
148                     type = "null category";
149             }
150             String averageGrad = Double.toString(this.
                    segmentAverageGradient);
151             String length = Double.toString(this.segmentLength);
152             return String.format("Segment[%s]: %s; %skm; Location=%
                    s; Gradient=%s;",
153                                  id, type, length, location,
                                         averageGrad);
154         }
155 
156         /**
157          * @param id The ID of the segment
158          * @return A string representation of the segment instance
159          * @throws IDNotRecognisedException If no segment exists
                     with the requested
160          *                                     ID
161          */
162         public static String toString(int id) throws
                IDNotRecognisedException {
163             return getSegment(id).toString();
164         }
165 
166         /**
167          * @return The integer segmentId for the segment instance
168          */
169         public int getSegmentId() { return this.segmentId; }
170 
171         /**
172          * @return The integer representing the location of the
```

```
            segment instance
173         */
174        public double getSegmentLocation() { return this.
               segmentLocation; }
175
176        /**
177         * @param id The ID of the segment
178         * @return The integer representing the location of the
                segment instance
179         * @throws IDNotRecognisedException If no segment exists
                with the requested
180         *                                      ID
181         */
182        public static double getSegmentLocation(int id) throws
183                                                    IDNotRecognisedException
                                                        {
184            return getSegment(id).segmentLocation;
185        }
186
187        /**
188         * @return The type of the segment instance
189         */
190        public SegmentType getSegmentType() { return this.
               segmentType; }
191
192        /**
193         * @param id The ID of the segment
194         * @return The type of the segment instance
195         * @throws IDNotRecognisedException If no segment exists
                with the requested
196         *                                      ID
197         */
198        public static SegmentType getSegmentType(int id) throws
199                                                      IDNotRecognisedException
                                                          {
200            return getSegment(id).segmentType;
201        }
202
203        /**
204         * @return The average gradient of the segment instance
205         */
206        public double getSegmentAverageGradient() {
207            return this.segmentAverageGradient;
208        }
209
210        /**
211         * @param id The ID of the segment
212         * @return The average gradient of the segment instance
213         * @throws IDNotRecognisedException If no segment exists
                with the requested
```

```
214         *                                    ID
215         */
216        public static double getSegmentAverageGradient(int id)
               throws
217                                                          IDNotRecognisedException
                                                                 {
218            return getSegment(id).segmentAverageGradient;
219        }
220
221        /**
222         * @return The length of the segment instance
223         */
224        public double getSegmentLength() { return this.
               segmentLength; }
225
226        /**
227         * @param id The ID of the segment
228         * @return The length of the segment instance
229         * @throws IDNotRecognisedException If no segment exists
                with the requested
230         *                                    ID
231         */
232        public static double getSegmentLength(int id) throws
               IDNotRecognisedException {
233            return getSegment(id).segmentLength;
234        }
235
236        /**
237         * @param location The new location for the segment
                instance
238         */
239        public void setSegmentLocation(double location) {
240            this.segmentLength = location;
241        }
242
243        /**
244         * @param id The ID of the segment to be updated
245         * @param location The new location for the segment
                instance
246         * @throws IDNotRecognisedException If no segment exists
                with the requested
247         *                                    ID
248         */
249        public static void setSegmentLocation(int id, double
               location) throws
250                                                IDNotRecognisedException
                                                       {
251            getSegment(id).setSegmentLocation(location);
252        }
253
```

```
254     /**
255      * @param type The new type for the segment instance
256      */
257     public void setSegmentType(SegmentType type) {
258         this.segmentType = type;
259     }
260 
261     /**
262      * @param id The ID of the segment to be updated
263      * @param type The new type for the segment instance
264      * @throws IDNotRecognisedException If no segment exists
             with the requested
265      *                                     ID
266      */
267     public static void setSegmentType(int id, SegmentType type)
         throws
268                                       IDNotRecognisedException
                                             {
269         getSegment(id).setSegmentType(type);
270     }
271 
272     /**
273      * @param averageGradient The new average gradient for the
             segment instance
274      */
275     public void setSegmentAverageGradient(double
        averageGradient) {
276         this.segmentAverageGradient = averageGradient;
277     }
278 
279     /**
280      * @param id The ID of the segment to be updated
281      * @param averageGradient The new average gradient for the
             segment instance
282      * @throws IDNotRecognisedException If no segment exists
             with the requested
283      *                                     ID
284      */
285     public static void setSegmentAverageGradient(int id, double
         averageGradient)
286                                                     throws
                                                         IDNotRecognisedException
                                                          {
287         getSegment(id).setSegmentAverageGradient(
                averageGradient);
288     }
289 
290     /**
291      * @param length The new length for the segment instance
292      */
```

```
293     public void setSegmentLength(double length) {
294         this.segmentLength = length;
295     }
296 
297     /**
298      * @param id The ID of the segment to be updated
299      * @param length The new length for the segment instance
300      * @throws IDNotRecognisedException If no segment exists
             with the requested
301      *                                     ID
302      */
303     public static void setSegmentLength(int id, double length)
         throws
304                                         IDNotRecognisedException
                                                {
305         getSegment(id).setSegmentLength(length);
306     }
307 }
```

6 Result.java

```
1  package cycling;
2 
3  import java.util.ArrayList;
4  import java.util.Arrays;
5  import java.io.Serializable;
6  import java.time.LocalTime;
7  import java.time.format.DateTimeFormatter;
8  import java.time.temporal.ChronoUnit;
9 
10 /**
11  * Result encapsulates rider results per stage, and handles
       time adjustments and
12  * rankings (scoring is done externally based on points
       distributions defined in
13  * Cycling Portal)
14  *
15  * @author Thomas Newbold
16  * @version 1.1
17  */
18 public class Result implements Serializable {
19     // Static class attributes
20     public static ArrayList<Result> allResults = new ArrayList<
           Result>();
21 
22     /**
23      * @param stageId The ID of the stage
24      * @return An array of all results for a stage
25      */
```

```
26      public static Result[] getResultsInStage(int stageId) {
27          ArrayList<Result> stage = new ArrayList<Result>();
28          for(Result r : allResults) {
29              stage.add(r);
30          }
31          stage.removeIf(r -> r.getStageId()!=stageId);
32          Result[] resultsForStage = new Result[stage.size()];
33          for(int i=0; i<stage.size(); i++) {
34              resultsForStage[i] = stage.get(i);
35          }
36          return resultsForStage;
37      }
38
39      /**
40       * @param riderId The ID of the driver
41       * @return An array of all results for a driver
42       */
43      public static Result[] getResultsForRider(int riderId) {
44          ArrayList<Result> rider = new ArrayList<Result>(
                allResults);
45          rider.removeIf(r -> r.getRiderId()!=riderId);
46          Result[] resultsForRider = new Result[rider.size()];
47          for(int i=0; i<rider.size(); i++) {
48              resultsForRider[i] = rider.get(i);
49          }
50          return resultsForRider;
51      }
52
53      // Instance attributes
54      private int stageId;
55      private int riderId;
56      private LocalTime[] checkpoints;
57
58      /**
59       * Result constructor; creates a new result entry and adds
             to the
60       * allResults array.
61       *
62       * @param sId The ID of the stage the result refers to
63       * @param rId The ID of the rider who achieved the result
64       * @param check An array of times at which the rider
             reached each
65       *              checkpoint (including start and finish)
66       */
67      public Result(int sId, int rId, LocalTime... check) {
68          this.stageId = sId;
69          this.riderId = rId;
70          this.checkpoints = check;
71          Result.allResults.add(this);
72      }
```

```
73
74      /**
75       * @return A string representation of the Result instance
76       */
77      public String toString() {
78          String sId = Integer.toString(this.stageId);
79          String rId = Integer.toString(this.riderId);
80          int l = this.getCheckpoints().length;
81          String times[] = new String[l];
82          DateTimeFormatter formatter = DateTimeFormatter.
                ofPattern("HH:mm:ss");
83          for(int i=0; i<l; i++) {
84              times[i] = this.getCheckpoints()[i].format(
                    formatter);
85          }
86          return String.format("Stage[%s]-Rider[%s]: SplitTimes=%
                s", sId, rId, Arrays.toString(times));
87      }
88
89      /**
90       * @param sId The ID of the stage of the result instance
91       * @param rId The ID of the associated rider to the result
             instance
92       * @return The Result instance
93       * @throws IDNotRecognisedException If an instance for the
             rider/stage
94       *                                  combination is not
             found in the
95       *                                  allResults array
96       */
97      public static Result getResult(int sId, int rId) throws
            IDNotRecognisedException {
98          for(Result r : allResults) {
99              if(r.getRiderId()==rId && r.getStageId()==sId) {
100                 return r;
101             }
102         }
103         throw new IDNotRecognisedException("results not found
                for rider in stage");
104     }
105
106     /**
107      * @param sId The ID of the stage of the result instance to
             remove
108      * @param rId The ID of the associated rider to the result
             instance to remove
109      * @throws IDNotRecognisedException If an instance for the
             rider/stage
110      *                                  combination is not
             found in the
```

```
111         *                                   allResults array
112         */
113         public static void removeResult(int sId, int rId) throws
                IDNotRecognisedException {
114             for(Result r : allResults) {
115                 if(r.getRiderId()==rId && r.getStageId()==sId) {
116                     allResults.remove(r);
117                     break;
118                 }
119             }
120             throw new IDNotRecognisedException("results not found
                    for rider in stage");
121         }
122
123         /**
124          * @return The stageId of the stage the result refers to
125          */
126         public int getStageId() { return this.stageId; }
127
128         /**
129          * @return The riderId of the rider associated with the
                result
130          */
131         public int getRiderId() { return this.riderId; }
132
133         /**
134          * @return An array of the split times between each
                checkpoint
135          */
136         public LocalTime[] getCheckpoints() {
137             LocalTime[] out = new LocalTime[this.checkpoints.length
                    -1];
138             for(int n=0;n<this.checkpoints.length-1; n++) {
139                 out[n] = getElapsed(checkpoints[n],checkpoints[n
                        +1]);
140             }
141             return out;
142         }
143
144         /**
145          * @return The total time elapsed between the start and end
                checkpoints
146          */
147         public LocalTime getTotalElasped() {
148             return Result.getElapsed(this.checkpoints[0], this.
                    checkpoints[-1]);
149         }
150
151         /**
152          * @param a Start time
```

```
153         * @param b End time
154         * @return The time difference between two times, a and b
155         */
156        public static LocalTime getElapsed(LocalTime a, LocalTime b
               ) {
157            int hours = (int)a.until(b, ChronoUnit.HOURS);
158            int minuites = (int)a.until(b, ChronoUnit.MINUTES);
159            int seconds = (int)a.until(b, ChronoUnit.SECONDS);
160            return LocalTime.of(hours%24, minuites%60, seconds%60);
161        }
162
163        /**
164         * @return An array of the checkpoint times, adjusted to a
                threshold of
165         *         one second
166         */
167        public LocalTime[] adjustedCheckpoints() {
168            LocalTime[] adjusted = this.getCheckpoints();
169            for(int n=0; n<adjusted.length; n++) {
170                adjusted[n] = adjustedCheckpoint(n);
171            }
172            return adjusted;
173        }
174
175        /**
176         * Recursive adjuster, used in {@link #adjustedCheckpoints
                ()}.
177         *
178         * @param n The index of the checkpoint to adjust
179         * @return The adjusted time for checkpoint n
180         */
181        public LocalTime adjustedCheckpoint(int n) {
182            for(int i=0; i<allResults.size(); i++) {
183                Result r = allResults.get(i);
184                if(r.getRiderId()==this.getRiderId() && r.
                       getStageId()==this.getStageId()) {
185                    continue;
186                }
187                LocalTime selfTime = this.getCheckpoints()[n];
188                LocalTime rTime = r.getCheckpoints()[n];
189                if(selfTime.until(rTime, ChronoUnit.SECONDS)<1) {
190                    return r.adjustedCheckpoint(n);
191                } else {
192                    return selfTime;
193                }
194            }
195            return null;
196        }
197    }
```

7 Team.java

```
1  package cycling;
2  import java.io.Serializable;
3  import java.util.ArrayList;
4  /**
5   * Team Class holds the teamId,name,description and riderIds
        belonging to that team.
6   *
7   *
8   * @author Ethan Ray
9   * @version 1.0
10  *
11  */
12
13 public class Team implements Serializable {
14      public static ArrayList<String> teamNames = new ArrayList
            <>();
15      public static int teamTopId = 0;
16
17      private int teamID;
18      private String name;
19      private String description;
20      private ArrayList<Integer> riderIds = new ArrayList<>();
21
22
23      /**
24       * @param name String - A name for the team, , If the name
             is null, empty, has more than 30 characters, or has
             white spaces will throw InvaildNameException.
25       * @param description String - A description for the team.
26       * @throws IllegalNameException name String - Is a
             duplicate name of any other Team, IllegalNameException
             will be thrown.
27       * @throws InvailNameException name String - If the name is
              null, empty, has more than 30 characters, or has white
              spaces will throw InvaildNameException.
28       */
29      public Team(String name, String description) throws
            IllegalNameException, InvalidNameException
30      {
31          if (name == "" || name.length()>30 || name.contains(" "
                )){
32              throw new InvalidNameException("Team name cannot be
                     empty, longer than 30 characters , or has white
                     spaces.");
33          }
34          for (int i = 0;i<teamNames.size();i++){
35              if (teamNames.get(i) == name){
```

```
36                  throw new IllegalNameException("That team name
                        already exsists!");
37              }
38          }
39
40          teamNames.add(name);
41          this.teamID = teamTopId++;
42          this.name = name;
43          this.description = description;
44      }
45      /**
46       * @param rider Rider - A rider to add to the team.
47       */
48      public void addRider(Rider rider){
49
50          this.riderIds.add(rider.getRiderId());
51      }
52      /**
53       * @param riderId int - A riderId to be removed from the
             team.
54       */
55      public void removeRiderId(int riderId){
56          for (int i =0;i<this.riderIds.size();i++){
57              if (this.riderIds.get(i)==riderId){
58                  this.riderIds.remove(i);
59                  break;
60              }
61          }
62      }
63      /**
64       * @return An Array of integers - which are the riderIds in
              that team.
65       */
66      public int[] getRiderIds(){
67          int [] currentRiderIds = new int[this.riderIds.size()];
68          for (int i=0; i<this.riderIds.size();i++){
69              currentRiderIds[i]=this.riderIds.get(i);
70          }
71          return currentRiderIds;
72      }
73      /**
74       * @return A Integer - teamId of the team.
75       */
76      public int getId(){
77          return this.teamID;
78      }
79      /**
80       * @return A String - Name of the team.
81       */
82      public String getTeamName(){
```

```
83          return this.name;
84      }
85      /**
86       * @return A String - The description of the team.
87       */
88      public String getDescription(){
89          return this.description;
90      }
91  }
```

8 Rider.java

```
1   package cycling;
2
3   import java.io.Serializable;
4
5   /**
6    * Rider Class holds the riders teamId,riderId,name and
         yearOfBirth
7    *
8    *
9    * @author Ethan Ray
10   * @version 1.0
11   *
12   */
13
14
15  public class Rider implements Serializable {
16      public static int ridersTopId;
17      private int riderId;
18      private int teamID;
19      private String name;
20      private int yearOfBirth;
21
22
23      /**
24       * @param teamID int - A team Id that the rider will belong
             too
25       * @param name String - A name for the rider, Has to be non
             -null or IllegalArgumentException is thrown.
26       * @param yearOfBirth int - A year that the rider was born
             in. Has to be above 1900 or IllegalArgumentException is
             thrown.
27       * @throws IllegalArgumentException name String - Has to be
             non-null or IllegalArgumentException is thrown.
28       * @throws IllegalArgumentException yearOfBirth int - A
             year that the rider was born in. Has to be above 1900
             or IllegalArgumentException is thrown.
29       */
```

```
30      public Rider(int teamID, String name, int yearOfBirth)
            throws IllegalArgumentException
31      {
32          this.riderId = ridersTopId++;
33          this.teamID = teamID;
34          if (name == "" || name == null){
35              throw new IllegalArgumentException("Illegal name
                    entered for rider");
36          }
37          this.name = name;
38          if (yearOfBirth < 1900){
39              throw new IllegalArgumentException("Illegal value
                    for yearOfBirth given please enter a value above
                     1900.");
40          }
41          this.yearOfBirth = yearOfBirth;
42      }
43      /**
44       * @return The RiderId of the rider.
45       */
46      public int getRiderId(){
47          return this.riderId;
48      }
49      /**
50       * @return The team Id that the rider belongs to/
51       */
52      public int getRiderTeamId(){
53          return this.teamID;
54      }
55      /**
56       * @return The rider's name.
57       */
58      public String getRiderName(){
59          return this.name;
60      }
61      /**
62       * @return The the year of birth of the rider.
63       */
64      public int getRiderYOB(){
65          return this.yearOfBirth;
66      }
67
68  }
```

9 RiderManager.java

```
1   package cycling;
2
3   import java.io.Serializable;
```

```
4  import java.util.ArrayList;
5
6  public class RiderManager implements Serializable{
7      public static ArrayList<Rider> allRiders = new ArrayList
           <>();
8      public static ArrayList<Team> allTeams = new ArrayList<>();
9
10
11     /**
12      * @param teamID int - A team Id that the rider will belong
              too. If the ID doesn't exist IDNotRecognisedException
             is thrown.
13      * @param name String - A name for the rider, Has to be non
             -null or IllegalArgumentException is thrown.
14      * @param yearOfBirth int - A year that the rider was born
             in. Has to be above 1900 or IllegalArgumentException is
              thrown.
15      * @return riderId of the rider created.
16      * @throws IDNotRecognisedException teamId int - If the ID
             doesn't exist IDNotRecognisedException is thrown.
17      * @throws IllegalArgumentException yearOfBirth int - A
             year that the rider was born in. Has to be above 1900
             or IllegalArgumentException is thrown.
18      */
19     int createRider(int teamID, String name, int yearOfBirth)
           throws IDNotRecognisedException,IllegalArgumentException
           {
20         int teamIndex = getIndexForTeamId(teamID);
21         Rider newRider = new Rider(teamID,name,yearOfBirth);
22         allRiders.add(newRider);
23         Team ridersTeam = allTeams.get(teamIndex);
24         ridersTeam.addRider(newRider);
25         return newRider.getRiderId();
26     }
27     /**
28      * @param riderId int - A riderId of a rider to be removed.
              If the ID doesn't exist IDNotRecognisedException is
             thrown.
29      * @throws IDNotRecognisedException riderId int - If the ID
              doesn't exist IDNotRecognisedException is thrown.
30      */
31     void removeRider(int riderId) throws
           IDNotRecognisedException
32     {
33         int riderIndex = getIndexForRiderId(riderId);
34         int teamId = allRiders.get(riderIndex).getRiderTeamId()
               ;
35         int teamIndex = getIndexForTeamId(teamId);
36         Team riderTeam = allTeams.get(teamIndex);
37         riderTeam.removeRiderId(riderId);
```

```
38          allRiders.remove(riderIndex);
39      }
40      /**
41       * @param riderId int - A riderId of a rider to be searced
             for. If the ID doesn't exist IDNotRecognisedException
             is thrown.
42       * @throws IDNotRecognisedException riderId int - If the ID
             doesn't exist IDNotRecognisedException is thrown.
43       * @return An int which is the index that maps to the
             riderId.
44       */
45      int getIndexForRiderId(int riderId) throws
            IDNotRecognisedException{
46          int index =-1;
47          if (allRiders.size() == 0){
48              throw new IDNotRecognisedException("No rider exists
                    with that ID");
49          }
50          for (int i=0; i<allRiders.size();i++){
51              if (allRiders.get(i).getRiderId()==riderId){
52                  index = i;
53                  break;
54              }
55          }
56          if (index == -1){
57              throw new IDNotRecognisedException("No rider exists
                    with that ID");
58          }
59          return index;
60      }
61      /**
62       * @param name String - A name for the team, , If the name
             is null, empty, has more than 30 characters, or has
             white spaces will throw InvaildNameException.
63       * @param description String - A description for the team.
64       * @throws IllegalNameException name String - Is a
             duplicate name of any other Team, IllegalNameException
             will be thrown.
65       * @throws InvailNameException name String - If the name is
             null, empty, has more than 30 characters, or has white
             spaces will throw InvaildNameException.
66       */
67      int createTeam(String name, String description) throws
            IllegalNameException, InvalidNameException{
68          Team newTeam = new Team(name,description);
69          allTeams.add(newTeam);
70          return newTeam.getId();
71      }
72      /**
73       * @param teamId int - A teamId of a rider to be removed.
```

```
74          If the ID doesn't exist IDNotRecognisedException is thrown.
75          * @throws IDNotRecognisedException riderId int - If the ID doesn't exist IDNotRecognisedException is thrown.
76          */
77         void removeTeam(int teamId) throws IDNotRecognisedException { // Delete team and all riders in that team
78             int teamIndex = getIndexForTeamId(teamId);
79             Team currentTeam = allTeams.get(teamIndex);
80             for (Integer riderId : currentTeam.getRiderIds()) {
81                 removeRider(riderId);
82             }
83             allTeams.remove(teamIndex);
84 
85         }
86         /**
87          * @return All the teamId's that are currently in the system as an int[]
88          */
89 
90         int[] getTeams(){
91             int [] allTeamIds = new int[allTeams.size()];
92             for (int i=0; i<allTeams.size();i++){
93                 allTeamIds[i]=allTeams.get(i).getId();
94             }
95             return allTeamIds;
96         }
97         /**
98          * @param teamId int - A teamId to get RidersId in that team. If the ID doesn't exist IDNotRecognisedException is thrown.
99          * @throws IDNotRecognisedException teamId int - If the ID doesn't exist IDNotRecognisedException is thrown.
100         * @return All the riderId's in a team as an int[]
101         */
102        int[] getTeamRiders(int teamId) throws IDNotRecognisedException{
103            Team currentTeam = getTeam(teamId);
104            return currentTeam.getRiderIds();
105 
106        }
107        /**
108         * @return All team names in the system as an String[]
109         */
110        String[] getTeamsNames(){
111            String [] allTeamNames = new String[allTeams.size()];
112            for (int i=0; i<allTeams.size();i++){
113                allTeamNames[i] = allTeams.get(i).getTeamName();
114            }
            return allTeamNames;
```

```
115     }
116     /**
117      * @return All rider names in the system as an String[]
118      */
119     String[] getRidersNames(){
120         String [] allRiderNames = new String[allRiders.size()];
121         for (int i=0; i<allRiders.size();i++){
122             allRiderNames[i] = allRiders.get(i).getRiderName();
123         }
124         return allRiderNames;
125     }
126     /**
127      * @param teamId int - A teamId of a team to search for its
           index. If the ID doesn't exist
          IDNotRecognisedException is thrown.
128      * @throws IDNotRecognisedException teamId int - If the ID
          doesn't exist IDNotRecognisedException is thrown.
129      * @return An int which is the index that maps to the
          teamId.
130      */
131     int getIndexForTeamId(int teamId) throws
         IDNotRecognisedException{
132         int index =-1;
133         if (allTeams.size() == 0){
134             throw new IDNotRecognisedException("No Team exists
                with that ID");
135         }
136         for (int i=0; i<allTeams.size();i++){
137             if (allTeams.get(i).getId()==teamId){
138                 index = i;
139                 break;
140             }
141         }
142         if (index == -1){
143             throw new IDNotRecognisedException("No rider exists
                with that ID");
144         }
145         return index;
146     }
147     /**
148      * @param teamId int - A teamId of a team to search for its
           object. If the ID doesn't exist
          IDNotRecognisedException is thrown.
149      * @throws IDNotRecognisedException teamId int - If the ID
          doesn't exist IDNotRecognisedException is thrown.
150      * @return A Team object with the teamId parsed.
151      */
152     Team getTeam(int teamId) throws IDNotRecognisedException{
153         int teamIndex = getIndexForTeamId(teamId);
154         return allTeams.get(teamIndex);
```

```
155     }
156     /**
157      * @param riderId int - A riderId of a team to search for
             its object. If the ID doesn't exist
             IDNotRecognisedException is thrown.
158      * @throws IDNotRecognisedException riderId int - If the ID
             doesn't exist IDNotRecognisedException is thrown.
159      * @return A Rider object with the riderId parsed.
160      */
161     Rider getRider(int riderId) throws IDNotRecognisedException
            {
162         int riderIndex = getIndexForRiderId(riderId);
163         return allRiders.get(riderIndex);
164     }
165     void setAllTeams(ArrayList<Team> allTeams){
166
167         RiderManager.allTeams = allTeams;
168         if (allTeams.size() != 0){
169         Team lastTeam = allTeams.get(allTeams.size()-1);
170         Team.teamTopId = lastTeam.getId()+1;
171         }
172     }
173     void setAllRiders(ArrayList<Rider> allRiders){
174         RiderManager.allRiders = allRiders;
175         if (allRiders.size() != 0){
176             Rider lastRider = allRiders.get(allRiders.size()-1)
                    ;
177             Rider.ridersTopId = lastRider.getRiderId()+1;
178         }
179     }
180
181 }
```