

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
18  * methods for
19  * handling the following classes: Race, Stage, Segment,
20  * RiderManager (and in
21  * turn Rider and Team), and Result.
22  * These classes are used manage races and their subdivisions,
23  * teams and their
24  * riders, and to calculate and assign points.
25  * Also contains methods for saving and loading
26  * MiniCyclingPortalInterface to
27  * and from a file.
28  *
29  * @author Ethan Ray & Thomas Newbold
30  * @version 1.0
31  */
32 public class CyclingPortal implements CyclingPortalInterface {
33     public RiderManager riderManager = new RiderManager();
34
35     @Override
36     public int[] getRaceIds() {
37         return Race.getAllRaceIds();
38     }
39
40     @Override
41     public int createRace(String name, String description)
42         throws IllegalArgumentException, InvalidNameException {
43         Race r = new Race(name, description);
44         return r.getRaceId();
45     }
46 }
```

```

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
87         location, SegmentType type, Double averageGradient,
88         Double length) throws IDNotRecognisedException,
89         InvalidLocationException,
90         InvalidStageStateException,
91         InvalidStageTypeException {
92         return Stage.addSegmentToStage(stageId, location, type,
93             averageGradient, length);
94     }
95
96     @Override
97     public int addIntermediateSprintToStage(int stageId, double
98         location) throws IDNotRecognisedException,
99         InvalidLocationException,
100         InvalidStageStateException,
101         InvalidStageTypeException {
102         // TODO Check inputs?
103         return Stage.addSegmentToStage(stageId, location,
104             SegmentType.SPRINT, 0.0, location);
105     }
106
107     @Override
108     public void removeSegment(int segmentId) throws
109         IDNotRecognisedException, InvalidStageStateException {
110         Stage.removeSegment(segmentId);
111     }
112
113     @Override
114     public void concludeStagePreparation(int stageId) throws
115         IDNotRecognisedException, InvalidStageStateException {
116         Stage.updateStageState(stageId);
117     }
118
119     @Override
120     public int[] getStageSegments(int stageId) throws
121         IDNotRecognisedException {
122         return Stage.getSegments(stageId);
123     }
124
125     @Override
126     public int createTeam(String name, String description)
127         throws IllegalNameException, InvalidNameException {
128         return riderManager.createTeam(name, description);
129     }
130
131     @Override

```

```

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

```

```

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.getTotalElapsed();
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

```

```

200         riderId) throws IDNotRecognisedException {
201             Stage.getStage(stageId);
202             riderManager.getRider(riderId);
203             // above should throw exceptions if IDs are not in
204             // system
205             Result.removeResult(stageId, riderId);
206         }
207     }
208
209     @Override
210     public int[] getRidersRankInStage(int stageId) throws
211         IDNotRecognisedException {
212         Result[] results = Result.getResultsInStage(stageId);
213         int[] riderRanks = new int[results.length];
214         Arrays.fill(riderRanks, -1);
215         for(Result r : results) {
216             for(int i=0; i<riderRanks.length; i++) {
217                 if(riderRanks[i] == -1) {
218                     riderRanks[i] = r.getRiderId();
219                 } else {
220                     if(r.getTotalElapsed().isBefore(Result.
221                         getResult(stageId, riderRanks[i]).
222                         getTotalElapsed())) {
223                         int temp;
224                         int prev = r.getRiderId();
225                         for(int j=i; j<riderRanks.length; j++)
226                         {
227                             temp = riderRanks[j];
228                             riderRanks[j] = prev;
229                             prev = temp;
230                             if(prev == -1) {
231                                 break;
232                             }
233                         }
234                     }
235                     break;
236                 }
237             }
238         }
239         return riderRanks;
240     }
241
242     @Override
243     public LocalTime[] getRankedAdjustedElapsedTimesInStage(int
244         stageId) throws IDNotRecognisedException {
245         // TODO Auto-generated method stub
246         // TODO Thomas do this after mountain points
247         return null;
248     }
249
250     @Override

```

```

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

```

```

280         segment
281     switch(type) {
282         case C4:
283             distribution = new int[]{1};
284             break;
285         case C3:
286             distribution = new int[]{2,1};
287             break;
288         case C2:
289             distribution = new int[]{5,3,2,1};
290             break;
291         case C1:
292             distribution = new int[]{10,8,6,4,2,1};
293             break;
294         case HC:
295             distribution = new int
296                 []{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(
309                     stageId, riderRanks[i]);
310                 if(r.getCheckpoints()[s].isBefore(
311                     compare.getCheckpoints()[s])) {
312                     int temp;
313                     int prev = r.getRiderId();
314                     for(int j=i; j<riderRanks.length; j
315                         ++){
316                         temp = riderRanks[j];
317                         riderRanks[j] = prev;
318                         prev = temp;
319                         if(prev == -1) {
320                             break;
321                         }
322                     }
323                 }
324             }
325         }
326     }
327     //return riderRanks;
328     ArrayList<Integer> ridersArray = new ArrayList<

```



```

Integer>());
325     for(int r : riders) { ridersArray.add(r); }
326     for(int i=0; i<Math.min(points.length, distribution
        .length); i++) {
327         int overallPos = ridersArray.indexOf(ridersRanks
            [i]);
328         if(overallPos<points.length && overallPos!=-1)
            {
329             points[overallPos] += distribution[i];
330         }
331     }
332 }
333 return points;
334 }
335
336 @Override
337 public void eraseCyclingPortal() {
338
339     Team.teamNames.clear();
340     Team.teamTopId = 0;
341     Rider.ridersTopId = 0;
342
343     RiderManager.allRiders.clear();
344     RiderManager.allTeams.clear();
345
346
347     Race.allRaces.clear();
348     Race.removedIds.clear();
349     Race.loadId();
350
351     Segment.allSegments.clear();
352     Segment.removedIds.clear();
353     Segment.loadId();
354
355     Stage.allStages.clear();
356     Stage.removedIds.clear();
357     Stage.loadId();
358
359     Result.allResults.clear();
360
361 }
362
363
364 @Override
365 public void saveCyclingPortal(String filename) throws
    IOException {
366     try {
367         FileOutputStream fos = new FileOutputStream(
            filename);
368         ObjectOutputStream oos = new ObjectOutputStream(fos

```

```

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

```

```

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

```

```

462         }
463         if (obj.getClass() == Segment.class){
464             Segment newSeg = (Segment) obj;
465             allSegments.add(newSeg);
466             System.out.println("NEW SEGMENT");
467             classFlag = Segment.class;
468         }
469
470
471         System.out.println(obj.getClass());
472     }
473 }
474 if (classFlag == Race.class){
475     Race.removedIds = removedIds;
476 }
477 if (classFlag == Segment.class){
478     Segment.removedIds = removedIds;
479 }
480 if (classFlag == Stage.class){
481     Stage.removedIds = removedIds;
482 }
483
484     this.riderManager.setAllTeams(allTeams);
485     this.riderManager.setAllRiders(allRiders);
486     Race.allRaces = allRaces;
487     Race.loadId();
488     Stage.allStages = allStages;
489     Stage.loadId();
490     Segment.allSegments = allSegments;
491     Segment.loadId();
492     Result.allResults = allResults;
493     ois.close();
494
495 }
496 catch (Exception ex) {
497     ex.printStackTrace();
498 }
499
500 }
501
502 @Override
503 public void removeRaceByName(String name) throws
504     NameNotRecognisedException {
505     boolean found = false;
506     for (int raceId : Race.getAllRaceIds()){ //Throwing
507         this exception is impossible!
508         try {
509             if (name == Race.getRaceName(raceId)){
510                 Race.removeRace(raceId);
511             }

```

```

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

```

```

553         return null;
554     }
555
556 }

```

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
9   * associated
10  * Stages.
11  *
12  * @author Thomas Newbold
13  * @version 2.0
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
19         <Integer>();
20     public static ArrayList<Race> allRaces = new ArrayList<Race
21         >();
22
23     /**
24      * Loads the value of idMax.
25      */
26     public static void loadId(){
27         if(Race.allRaces.size()!=0) {
28             Race.idMax = Race.allRaces.get(Race.allRaces.size()
29                 -1).getRaceId() + 1;
30         } else {
31             Race.idMax = 0;
32         }
33     }
34
35     /**
36      * @param raceId The ID of the race instance to fetch
37      * @return The race instance with the associated ID
38      * @throws IDNotRecognisedException If no race exists with
39      * the requested ID
40      */
41     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
49             instance for raceID");
50     } else {
51         throw new IDNotRecognisedException("raceID out of
52             range");
53     }
54 }
55
56 /**
57  * @return An integer array of the race IDs of all races
58  */
59 public static int[] getAllRaceIds() {
60     int length = Race.allRaces.size();
61     int[] raceIdsArray = new int[length];
62     int i = 0;
63     for(Race race : allRaces) {
64         raceIdsArray[i] = race.getRaceId();
65         i++;
66     }
67     return raceIdsArray;
68 }
69
70 /**
71  * @param raceId The ID of the race instance to remove
72  * @throws IDNotRecognisedException If no race exists with
73  *     the requested ID
74  */
75 public static void removeRace(int raceId) throws
76     IDNotRecognisedException {
77     boolean removed = Race.removedIds.contains(raceId);
78     if(raceId<Race.idMax && raceId >= 0 && !removed) {
79         Race r = getRace(raceId);
80         for(int id : r.getStages()) {
81             r.removeStageFromRace(id);
82         }
83         allRaces.remove(raceId);
84         removedIds.add(raceId);
85     } else if (removed) {
86         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 IllegalArgumentException 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
    IllegalArgumentException,
120     InvalidNameException {
121     for(Race race : allRaces) {
122         if(race.getRaceName().equals(name)) {
123             throw new IllegalArgumentException("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;",
150         id, name,
151         description, list);
152 }
153
154 /**
155  * @param id The ID of the race
156  * @return A string representation of the race instance
157  * @throws IDNotRecognisedException If no race exists with
158  *         the requested ID
159  */
160 public static String toString(int id) throws
161     IDNotRecognisedException {
162     return getRace(id).toString();
163 }
164
165 /**
166  * @return The integer raceId for the race instance
167  */
168 public int getRaceId() { return this.raceId; }
169
170 /**
171  * @return The string raceName for the race instance
172  */
173 public String getRaceName() { return this.raceName; }

```

```

172     /**
173      * @param id The ID of the race
174      * @return The string raceName for the race with the
175      *         associated id
176      * @throws IDNotRecognisedException If no race exists with
177      *         the requested ID
178      */
179     public static String getRaceName(int id) throws
180         IDNotRecognisedException {
181         return getRace(id).raceName;
182     }
183
184     /**
185      * @return The string raceDescription for the race instance
186      */
187     public String getRaceDescription() { return this.
188         raceDescription; }
189
190     /**
191      * @param id The ID of the race
192      * @return The string raceDescription for the race with the
193      *         associated id
194      * @throws IDNotRecognisedException If no race exists with
195      *         the requested ID
196      */
197     public static String getRaceDescription(int id) throws
198         IDNotRecognisedException
199     {
200         return getRace(id).raceDescription;
201     }
202
203     /**
204      * @return An integer array of stage IDs for the race
205      *         instance
206      */
207     public int[] getStages() {
208         int length = this.stageIds.size();
209         int[] stageIdsArray = new int[length];
210         for(int i=0; i<length; i++) {
211             stageIdsArray[i] = this.stageIds.get(i);
212         }
213         return stageIdsArray;
214     }
215
216     /**
217      * @param id The ID of the race
218      * @return An integer array of stage IDs for the race
219      *         instance
220      * @throws IDNotRecognisedException If no race exists with
221      *         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
234         the requested ID
235     */
236     public static void setRaceName(int id, String name) throws
        IDNotRecognisedException {
237         getRace(id).setRaceName(name);
238     }
239
240     /**
241     * @param description The new description for the race
242         instance
243     */
244     public void setRaceDescription(String description) {
245         this.raceDescription = description;
246     }
247
248     /**
249     * @param id The ID of the race to be updated
250     * @param description The new description for the race
251         instance
252     * @throws IDNotRecognisedException If no race exists with
253         the requested ID
254     */
255     public static void setRaceDescription(int id, String
        description) throws
        IDNotRecognisedException
        {
        getRace(id).setRaceDescription(description);

```

```

255     }
256
257     /**
258     * Creates a new stage and adds the ID to the stageIds
259     * array.
260     *
261     * @param name The name of the new stage
262     * @param description The description of the new stage
263     * @param length The length of the new stage (in km)
264     * @param startTime The date and time at which the stage
265     * will be held
266     * @param type The StageType, used to determine the point
267     * distribution
268     * @return The ID of the new stage
269     */
270     public int addStageToRace(String name, String description,
271                               double length,
272                               LocalDateTime startTime,
273                               StageType type) throws
274                               IllegalArgumentException,
275                               InvalidNameException,
276                               InvalidLengthException {
277         Stage newStage = new Stage(name, description, length,
278                                     startTime, type);
279         this.stageIds.add(newStage.getStageId());
280         return newStage.getStageId();
281     }
282
283     /**
284     * Creates a new stage and adds the ID to the stageIds
285     * array.
286     *
287     * @param id The ID of the race to which the stage will be
288     * added
289     * @param name The name of the new stage
290     * @param description The description of the new stage
291     * @param length The length of the new stage (in km)
292     * @param startTime The date and time at which the stage
293     * will be held
294     * @param type The StageType, used to determine the point
295     * distribution
296     * @return The ID of the new stage
297     * @throws IDNotRecognisedException If no race exists with
298     * the requested ID
299     */
300     public static int addStageToRace(int id, String name,
301                                     String description,
302                                     double length,
303                                     LocalDateTime startTime,
304                                     ,

```

```

290                                     StageType type) throws
291                                     IDNotRecognisedException,
292                                     IllegalNameException,
293                                     InvalidNameException,
294                                     InvalidLengthException {
295     return getRace(id).addStageToRace(name, description,
296                                     length, startTime, type);
297 }
298
299 /**
300  * Removes a stageId from the array of stageIds for a race
301  * instance,
302  * as well as from the static array of all stages in the
303  * Stage class.
304  *
305  * @param stageId The ID of the stage to be removed
306  * @throws IDNotRecognisedException If no stage exists with
307  * the requested ID
308  */
309 private void removeStageFromRace(int stageId) throws
310     IDNotRecognisedException {
311     if(this.stageIds.contains(stageId)) {
312         this.stageIds.remove(stageId);
313         Stage.removeStage(stageId);
314     } else {
315         throw new IDNotRecognisedException("stageID not
316             found in race");
317     }
318 }
319
320 /**
321  * Removes a stageId from the array of stageIds for a race
322  * instance,
323  * as well as from the static array of all stages in the
324  * Stage class.
325  *
326  * @param id The ID of the race to which the stage will be
327  * removed
328  * @param stageId The ID of the stage to be removed
329  * @throws IDNotRecognisedException If no stage exists with
330  * the requested ID
331  */
332 public static void removeStageFromRace(int id, int stageId)
333     throws
334     IDNotRecognisedException
335 {
336     getRace(id).removeStageFromRace(stageId);
337 }
338
339 /**

```

```

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.
28                 size()-1).getStageId() + 1;
29         } else {
30             Stage.idMax = 0;
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
37     *         the requested ID
38     */
39     public static Stage getStage(int stageId) throws
40         IDNotRecognisedException {
41         boolean removed = Stage.removedIds.contains(stageId);
42         if(stageId<Stage.idMax && stageId >= 0 && !removed) {
43             int index = stageId;
44             for(int j=0; j<Stage.removedIds.size(); j++) {
45                 if(Stage.removedIds.get(j) < stageId) {
46                     index--;
47                 }
48             }
49             return allStages.get(index);
50         } else if (removed) {
51             throw new IDNotRecognisedException("no stage
52                 instance for stageID");
53         } else {
54             throw new IDNotRecognisedException("stageId out of
55                 range");
56         }
57     }
58     /**
59     * @return An integer array of the stage IDs of all stage
60     */
61     public static int[] getAllStageIds() {
62         int length = Stage.allStages.size();
63         int[] stageIdsArray = new int[length];
64         int i = 0;
65         for(Stage stage : allStages) {
66             stageIdsArray[i] = stage.getStageId();
67             i++;
68         }
69         return stageIdsArray;
70     }
71     /**

```

```

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 IllegalArgumentException 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      IllegalArgumentException, InvalidNameException,
131      InvalidLengthException {
132      for(Stage stage : allStages) {
133      if(stage.getStageName().equals(name)) {
134      throw new IllegalArgumentException("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.
179             ofPattern("HH:hh dd-MM-yyyy");
180         String startTime = this.stageStartTime.format(formatter
181             );
182         String list = this.segmentIds.toString();
183         String type;
184         switch (this.stageType) {
185             case FLAT:
186                 type = "Flat";
187                 break;
188             case MEDIUM_MOUNTAIN:
189                 type = "Medium Mountain";
190                 break;
191             case HIGH_MOUNTAIN:
192                 type = "High Mountain";
193                 break;
194             case TT:
195                 type = "Time Trial";
196                 break;
197             default:
198                 type = "null type";
199         }
200         return String.format("Stage[%s] (%s): %s (%s); %s; %skm;
201             %s; SegmentIds=%s;",
202             id, state, name, type, description

```

```

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
           IDNotRecognisedException
228      {
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                                                     {
287         return getStage(id).stageStartTime;
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
298     *         the requested ID
299     */
300     public static StageType getStageType(int id) throws
301         IDNotRecognisedException {
302         return getStage(id).getStageType();
303     }
304
305     /**
306     * @return An integer array of segment IDs for the stage
307     *         instance
308     */
309     public int[] getSegments() {
310         int length = this.segmentIds.size();
311         int[] segmentIdsArray = new int[length];
312         for(int i=0; i<length; i++) {
313             segmentIdsArray[i] = this.segmentIds.get(i);
314         }
315         return segmentIdsArray;
316     }
317
318     /**
319     * @param id The ID of the stage
320     * @return An integer array of segment IDs for the stage
321     *         instance
322     * @throws IDNotRecognisedException If no stage exists with
323     *         the requested ID
324     */
325     public static int[] getSegments(int id) throws
326         IDNotRecognisedException {
327         Stage stage = getStage(id);
328         int length = stage.segmentIds.size();
329         int[] segmentIdsArray = new int[length];
330         for(int i=0; i<length; i++) {
331             segmentIdsArray[i] = stage.segmentIds.get(i);
332         }
333         return segmentIdsArray;

```

```

328     }
329
330     /**
331     * Updates the stage state from building to waiting for
332     * results.
333     *
334     * @throws InvalidStageStateException If the stage is
335     * already waiting for results
336     */
337     public void updateStageState() throws
338         InvalidStageStateException {
339         if(this.stageState.equals(StageState.WAITING)) {
340             throw new InvalidStageStateException("stage is
341             already waiting for results");
342         } else if(this.stageState.equals(StageState.BUILDING))
343         {
344             this.stageState = StageState.WAITING;
345         }
346     }
347
348     /**
349     * Updates the stage state from building to waiting for
350     * results.
351     *
352     * @param id The ID of the stage to be updated
353     * @throws IDNotRecognisedException If no stage exists with
354     * the requested ID
355     * @throws InvalidStageStateException If the stage is
356     * already waiting for results
357     */
358     public static void updateStageState(int id) throws
359         IDNotRecognisedException,
360         InvalidStageStateException
361     {
362         getStage(id).updateStageState();
363     }
364
365     /**
366     * @param name The new name for the stage instance
367     */
368     public void setStageName(String name) {
369         this.stageName = name;
370     }
371
372     /**
373     * @param id The ID of the stage to be updated
374     * @param name The new name for the stage instance
375     * @throws IDNotRecognisedException If no stage exists with
376     * the requested ID
377     */

```

```

367     public static void setStageName(int id, String name) throws
368                                     IDNotRecognisedException {
369         getStage(id).setStageName(name);
370     }
371
372     /**
373      * @param description The new description for the stage
374      * 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
383      * instance
384      * @throws IDNotRecognisedException If no stage exists with
385      * the requested ID
386      */
387     public static void setStageDescription(int id, String
388                                     description) throws
389                                     IDNotRecognisedException
390     {
391         getStage(id).setStageDescription(description);
392     }
393
394     /**
395      * @param length The new length for the stage instance
396      */
397     public void setStageLength(double length) {
398         this.stageLength = length;
399     }
400
401     /**
402      * @param id The ID of the stage to be updated
403      * @param length The new length for the stage instance
404      * @throws IDNotRecognisedException If no stage exists with
405      * the requested ID
406      */
407     public static void setStageLength(int id, double length)
408                                     throws
409                                     IDNotRecognisedException
410     {
411         getStage(id).stageLength = length;
412     }
413
414     /**
415      * @param startTime The new start time for the stage
416      * 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
416      * instance
417      * @throws IDNotRecognisedException If no stage exists with
418      * the requested ID
419      */
420     public static void setStageStartTime(int id, LocalDateTime
421         startTime)
422         throws
423         IDNotRecognisedException
424     {
425         getStage(id).stageStartTime = startTime;
426     }
427
428     /**
429      * Creates a new stage and adds the ID to the stageIds
430      * array.
431      *
432      * @param location The location of the new segment
433      * @param type The type of the new segment
434      * @param averageGradient The average gradient of the new
435      * segment
436      * @param length The length (in km) of the new segment
437      * @throws InvalidLocationException If the segment finishes
438      * outside of the
439      *
440      * bounds of the stage
441      * @throws InvalidStageStateException If the segment state
442      * is waiting for
443      *
444      * results
445      * @throws InvalidStageTypeException If the stage type is a
446      * time-trial
447      *
448      * (cannot contain
449      *
450      * segments)
451      */
452     public int addSegmentToStage(double location, SegmentType
453         type,
454         double averageGradient, double
455         length) throws
456         InvalidLocationException,
457         InvalidStageStateException,
458         InvalidStageTypeException {
459         if(location > this.getStageLength()) {
460             throw new InvalidLocationException("segment
461                 finishes outside of stage bounds");
462         }
463     }

```



```

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.getId());
453     return newSegment.getId();
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                                     {
479         return getStage(id).addSegmentToStage(location, type,
480         averageGradient, length);
481     }
482
483     /**
484     * Removes a segmentId from the array of segmentIds for a
485     * stage instance,
486     * as well as from the static array of all segments in the
487     * Segment class.
488     *
489     * @param segmentId The ID of the segment to be removed
490     * @throws IDNotRecognisedException If no segment exists
491     * with the requested
492     *
493     * ID
494     */
495     private void removeSegmentFromStage(int segmentId) throws
496     IDNotRecognisedException
497     {
498         if(this.segmentIds.contains(segmentId)) {
499             this.segmentIds.remove(segmentId);
500             Segment.removeSegment(segmentId);
501         } else {
502             throw new IDNotRecognisedException("segmentID not
503             found in race");
504         }
505     }
506
507     /**
508     * Removes a segmentId from the array of segmentIds for a
509     * stage instance,
510     * as well as from the static array of all segments in the
511     * Segment class.
512     *
513     * @param id The ID of the stage to which the segment will
514     * be removed
515     * @param segmentId The ID of the segment to be removed
516     * @throws IDNotRecognisedException If no segment exists
517     * with the requested
518     *
519     * ID
520     */
521     public static void removeSegmentFromStage(int id, int
522     segmentId) throws
523     IDNotRecognisedException
524     {
525         getStage(id).removeSegmentFromStage(segmentId);
526     }
527
528     /**

```

```

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
17         <Integer>();
18     public static ArrayList<Segment> allSegments = new
19         ArrayList<Segment>();
20
21     /**
22      * Loads the value of idMax.
23      */
24     public static void loadId() {
25         if (Segment.allSegments.size() != 0) {
26             Segment.idMax = Segment.allSegments.get(-1).
27                 getSegmentId() + 1;
28         } else {
29             Segment.idMax = 0;
30         }
31     }
32
33     /**
34      * @param segmentId The ID of the segment instance to fetch
35      * @return The segment instance with the associated ID
36      * @throws IDNotRecognisedException If no segment exists
37      *         with the requested
38      *
39      * ID
40      */
41     public static Segment getSegment(int segmentId) throws
42         IDNotRecognisedException {
43         boolean removed = Segment.removedIds.contains(segmentId
44             );
45         if (segmentId < Segment.idMax && segmentId >= 0 && !
46             removed) {
47             int index = segmentId;
48             for (int j=0; j<Segment.removedIds.size(); j++) {
49                 if (Segment.removedIds.get(j) < segmentId) {
50                     index--;
51                 }
52             }
53         }
54     }
55 }

```

```

45         }
46         return allSegments.get(index);
47     } else if (removed) {
48         throw new IDNotRecognisedException("no segment
49             instance for "+
49                                     "segmentId");
50     } else {
51         throw new IDNotRecognisedException("segmentId out
52             of range");
53     }
54 }
55 /**
56  * @return An integer array of the segment IDs of all
57  *         segment
58  */
59 public static int[] getAllSegmentIds() {
60     int length = Segment.allSegments.size();
61     int[] segmentIdsArray = new int[length];
62     int i = 0;
63     for(Segment segment : allSegments) {
64         segmentIdsArray[i] = segment.getSegmentId();
65         i++;
66     }
67     return segmentIdsArray;
68 }
69 /**
70  * @param segmentId The ID of the segment instance to
71  *         remove
72  * @throws IDNotRecognisedException If no segment exists
73  *         with the requested
74  *         ID
75  */
76 public static void removeSegment(int segmentId) throws
77     IDNotRecognisedException {
78     boolean removed = Segment.removedIds.contains(segmentId
79 );
80     if(segmentId<Segment.idMax && segmentId >= 0 && !
81         removed) {
82         allSegments.remove(segmentId);
83         Segment.idMax--;
84         for(int i=segmentId;i<allSegments.size();i++) {
85             getSegment(i).segmentId--;
86         }
87     } else if (removed) {
88         throw new IDNotRecognisedException("no segment
89             instance for "+
90                                     "segmentId");
91     } else {

```

```

87         throw new IDNotRecognisedException("segmentId out
88             of range");
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
100      * allSegment array.
101      *
102      * @param location The location of the finish of the new
103      * segment in the stage
104      * @param type The type of the new segment
105      * @param averageGradient The average gradient of the new
106      * segment
107      * @param length The length of the new segment
108      */
109     public Segment(double location, SegmentType type, double
110         averageGradient,
111         double length) {
112         if(Segment.removedIds.size() > 0) {
113             this.segmentId = Segment.removedIds.get(0);
114             Segment.removedIds.remove(0);
115         } else {
116             this.segmentId = idMax++;
117         }
118         this.segmentLocation = location;
119         this.segmentType = type;
120         this.segmentAverageGradient = averageGradient;
121         this.segmentLength = length;
122         Segment.allSegments.add(this);
123     }
124
125     /**
126      * @return A string representation of the segment instance
127      */
128     public String toString() {
129         String id = Integer.toString(this.segmentId);
130         String location = Double.toString(this.segmentLocation)
131             ;
132         String type;
133         switch (this.segmentType) {
134             case SPRINT:
135                 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 Catégorie";
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
160  *         with the requested 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
173  *         segment instance
174  */
175 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.segmentLocation = 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
265      *         with the requested ID
266      */
267     public static void setSegmentType(int id, SegmentType type)
268         throws IDNotRecognisedException
269     {
270         getSegment(id).setSegmentType(type);
271     }
272
273     /**
274      * @param averageGradient The new average gradient for the
275      *         segment instance
276      */
277     public void setSegmentAverageGradient(double
278         averageGradient) {
279         this.segmentAverageGradient = averageGradient;
280     }
281
282     /**
283      * @param id The ID of the segment to be updated
284      * @param averageGradient The new average gradient for the
285      *         segment instance
286      * @throws IDNotRecognisedException If no segment exists
287      *         with the requested ID
288      */
289     public static void setSegmentAverageGradient(int id, double
290         averageGradient)
291         throws IDNotRecognisedException
292     {
293         getSegment(id).setSegmentAverageGradient(
294             averageGradient);
295     }
296
297     /**
298      * @param length The new length for the segment instance
299      */
300     public void setSegmentLength(double length) {
301         this.segmentLength = length;
302     }

```

```

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
301     *                                   with the requested ID
302     */
303     public static void setSegmentLength(int id, double length)
304         throws IDNotRecognisedException
305     {
306         getSegment(id).setSegmentLength(length);
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.LocalDateTime;
7 import java.time.format.DateTimeFormatter;
8 import java.time.temporal.ChronoUnit;
9
10 /**
11  * Result encapsulates rider results per stage, and handles
12  * time adjustments and
13  * rankings (scoring is done externally based on points
14  * distributions defined in
15  * Cycling Portal)
16  *
17  * @author Thomas Newbold
18  * @version 1.1
19  */
20 public class Result implements Serializable {
21     // Static class attributes
22     public static ArrayList<Result> allResults = new ArrayList<
23         Result>();
24
25     /**
26     * @param stageId The ID of the stage
27     * @return An array of all results for a stage
28     */
29     public static Result[] getResultsInStage(int stageId) {
30         ArrayList<Result> stage = new ArrayList<Result>();
31         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>(
45         allResults);
46     rider.removeIf(r -> r.getRiderId() != riderId);
47     Result[] resultsForRider = new Result[rider.size()];
48     for(int i=0; i<rider.size(); i++) {
49         resultsForRider[i] = rider.get(i);
50     }
51     return resultsForRider;
52 }
53
54 // Instance attributes
55 private int stageId;
56 private int riderId;
57 private LocalTime[] checkpoints;
58
59 /**
60  * Result constructor; creates a new result entry and adds
61  * to the
62  * allResults array.
63  *
64  * @param sId The ID of the stage the result refers to
65  * @param rId The ID of the rider who achieved the result
66  * @param check An array of times at which the rider
67  * reached each
68  * checkpoint (including start and finish)
69  */
70 public Result(int sId, int rId, LocalTime... check) {
71     this.stageId = sId;
72     this.riderId = rId;
73     this.checkpoints = check;
74     Result.allResults.add(this);
75 }
76
77 /**
78  * @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

```

```

114         IDNotRecognisedException {
115             for(Result r : allResults) {
116                 if(r.getRiderId()==rId && r.getStageId()==sId) {
117                     allResults.remove(r);
118                     break;
119                 }
120             }
121             throw new IDNotRecognisedException("results not found
122                 for rider in stage");
123         }
124     /**
125      * @return The stageId of the stage the result refers to
126      */
127     public int getStageId() { return this.stageId; }
128
129     /**
130      * @return The riderId of the rider associated with the
131      *         result
132      */
133     public int getRiderId() { return this.riderId; }
134
135     /**
136      * @return An array of the split times between each
137      *         checkpoint
138      */
139     public LocalTime[] getCheckpoints() {
140         LocalTime[] out = new LocalTime[this.checkpoints.length
141             -1];
142         for(int n=0;n<this.checkpoints.length-1; n++) {
143             out[n] = getElapsed(checkpoints[n],checkpoints[n
144                 +1]);
145         }
146         return out;
147     }
148
149     /**
150      * @return The total time elapsed between the start and end
151      *         checkpoints
152      */
153     public LocalTime getTotalElapsed() {
154         LocalTime[] times = this.checkpoints;
155         return Result.getElapsed(times[0], times[times.length
156             -1]);
157     }
158
159     /**
160      * @param a Start time
161      * @param b End time
162      * @return The time difference between two times, a and b

```

```

156     */
157     public static LocalTime getElapsed(LocalTime a, LocalTime b
158         ) {
159         int hours = (int)a.until(b, ChronoUnit.HOURS);
160         int minutes = (int)a.until(b, ChronoUnit.MINUTES);
161         int seconds = (int)a.until(b, ChronoUnit.SECONDS);
162         return LocalTime.of(hours%24, minutes%60, seconds%60);
163     }
164
165     /**
166     * @return An array of the checkpoint times, adjusted to a
167     *         threshold of
168     *         one second
169     */
170     public LocalTime[] adjustedCheckpoints() {
171         LocalTime[] adjusted = this.getCheckpoints();
172         for(int n=0; n<adjusted.length; n++) {
173             adjusted[n] = adjustedCheckpoint(n);
174         }
175         return adjusted;
176     }
177
178     /**
179     * Recursive adjuster, used in {@link #adjustedCheckpoints
180     * ()}.
181     *
182     * @param n The index of the checkpoint to adjust
183     * @return The adjusted time for checkpoint n
184     */
185     public LocalTime adjustedCheckpoint(int n) {
186         for(int i=0; i<allResults.size(); i++) {
187             Result r = allResults.get(i);
188             if(r.getRiderId()==this.getRiderId() && r.
189                 getStageId()==this.getStageId()) {
190                 continue;
191             }
192             LocalTime selfTime = this.getCheckpoints()[n];
193             LocalTime rTime = r.getCheckpoints()[n];
194             if(selfTime.until(rTime, ChronoUnit.SECONDS)<1) {
195                 return r.adjustedCheckpoint(n);
196             } else {
197                 return selfTime;
198             }
199         }
200         return null;
201     }
202 }

```

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
6   * belonging to that team.
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         <>();
16     public static int teamTopId = 0;
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
25       * is null, empty, has more than 30 characters, or has
26       * white spaces will throw InvaildNameException.
27      * @param description String - A description for the team.
28      * @throws IllegalNameException name String - Is a
29       * duplicate name of any other Team, IllegalNameException
30       * will be thrown.
31      * @throws InvailNameException name String - If the name is
32       * null, empty, has more than 30 characters, or has white
33       * spaces will throw InvaildNameException.
34     */
35     public Team(String name, String description) throws
36         IllegalNameException, InvalidNameException
37     {
38         if (name == "" || name.length()>30 || name.contains(" ")
39             ){
40             throw new InvalidNameException("Team name cannot be
41                 empty, longer than 30 characters , or has white
42                 spaces.");
43         }
44         for (int i = 0;i<teamNames.size();i++){
45             if (teamNames.get(i) == name){
```



```

36         throw new IllegalArgumentException("That team name
37             already exists!");
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
54     team.
55  */
56 public void removeRiderId(int riderId){
57     for (int i =0;i<this.riderIds.size();i++){
58         if (this.riderIds.get(i)==riderId){
59             this.riderIds.remove(i);
60             break;
61         }
62     }
63 }
64 /**
65  * @return An Array of integers - which are the riderIds in
66     that team.
67  */
68 public int[] getRiderIds(){
69     int [] currentRiderIds = new int[this.riderIds.size()];
70     for (int i=0; i<this.riderIds.size();i++){
71         currentRiderIds[i]=this.riderIds.get(i);
72     }
73     return currentRiderIds;
74 }
75 /**
76  * @return A Integer - teamId of the team.
77  */
78 public int getId(){
79     return this.teamID;
80 }
81 /**
82  * @return A String - Name of the team.
83  */
84 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
7   * yearOfBirth
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
25       * too
26       * @param name String - A name for the rider, Has to be non
27       * -null or IllegalArgumentException is thrown.
28       * @param yearOfBirth int - A year that the rider was born
29       * in. Has to be above 1900 or IllegalArgumentException is
30       * thrown.
31       * @throws IllegalArgumentException name String - Has to be
32       * non-null or IllegalArgumentException is thrown.
33       * @throws IllegalArgumentException yearOfBirth int - A
34       * year that the rider was born in. Has to be above 1900
35       * or IllegalArgumentException is thrown.
36       */
37
38  }

```

```

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         <>();
9     public static ArrayList<Team> allTeams = new ArrayList<>();
10
11     /**
12      * @param teamID int - A team Id that the rider will belong
13      * too. If the ID doesn't exist IDNotRecognisedException
14      * is thrown.
15      * @param name String - A name for the rider, Has to be non
16      * -null or IllegalArgumentException is thrown.
17      * @param yearOfBirth int - A year that the rider was born
18      * in. Has to be above 1900 or IllegalArgumentException is
19      * thrown.
20      * @return riderId of the rider created.
21      * @throws IDNotRecognisedException teamId int - If the ID
22      * doesn't exist IDNotRecognisedException is thrown.
23      * @throws IllegalArgumentException yearOfBirth int - A
24      * year that the rider was born in. Has to be above 1900
25      * or IllegalArgumentException is thrown.
26      */
27     int createRider(int teamID, String name, int yearOfBirth)
28     throws IDNotRecognisedException,IllegalArgumentException
29     {
30         int teamIndex = getIndexForTeamId(teamID);
31         Rider newRider = new Rider(teamID,name,yearOfBirth);
32         allRiders.add(newRider);
33         Team ridersTeam = allTeams.get(teamIndex);
34         ridersTeam.addRider(newRider);
35         return newRider.getRiderId();
36     }
37     /**
38      * @param riderId int - A riderId of a rider to be removed.
39      * If the ID doesn't exist IDNotRecognisedException is
40      * thrown.
41      * @throws IDNotRecognisedException riderId int - If the ID
42      * doesn't exist IDNotRecognisedException is thrown.
43      */
44     void removeRider(int riderId) throws
45     IDNotRecognisedException
46     {
47         int riderIndex = getIndexForRiderId(riderId);
48         int teamId = allRiders.get(riderIndex).getRiderTeamId()
49         ;
50         int teamIndex = getIndexForTeamId(teamId);
51         Team riderTeam = allTeams.get(teamIndex);
52         riderTeam.removeRiderId(riderId);

```

```

38         allRiders.remove(riderIndex);
39     }
40     /**
41     * @param riderId int - A riderId of a rider to be searched
42     * for. If the ID doesn't exist IDNotRecognisedException
43     * is thrown.
44     * @throws IDNotRecognisedException riderId int - If the ID
45     * doesn't exist IDNotRecognisedException is thrown.
46     * @return An int which is the index that maps to the
47     * riderId.
48     */
49     int getIndexForRiderId(int riderId) throws
50     IDNotRecognisedException{
51         int index = -1;
52         if (allRiders.size() == 0){
53             throw new IDNotRecognisedException("No rider exists
54             with that ID");
55         }
56         for (int i=0; i<allRiders.size();i++){
57             if (allRiders.get(i).getRiderId()==riderId){
58                 index = i;
59                 break;
60             }
61         }
62         if (index == -1){
63             throw new IDNotRecognisedException("No rider exists
64             with that ID");
65         }
66         return index;
67     }
68     /**
69     * @param name String - A name for the team, , If the name
70     * is null, empty, has more than 30 characters, or has
71     * white spaces will throw InvaildNameException.
72     * @param description String - A description for the team.
73     * @throws IllegalNameException name String - Is a
74     * duplicate name of any other Team, IllegalNameException
75     * will be thrown.
76     * @throws InvailNameException name String - If the name is
77     * null, empty, has more than 30 characters, or has white
78     * spaces will throw InvaildNameException.
79     */
80     int createTeam(String name, String description) throws
81     IllegalNameException, InvalidNameException{
82         Team newTeam = new Team(name,description);
83         allTeams.add(newTeam);
84         return newTeam.getId();
85     }
86     /**
87     * @param teamId int - A teamId of a rider to be removed.

```

```

        If the ID doesn't exist IDNotRecognisedException is
        thrown.
74     * @throws IDNotRecognisedException riderId int - If the ID
        doesn't exist IDNotRecognisedException is thrown.
75     */
76     void removeTeam(int teamId) throws IDNotRecognisedException
        { // Delete team and all riders in that team
77         int teamIndex = getIndexForTeamId(teamId);
78         Team currentTeam = allTeams.get(teamIndex);
79         for (Integer riderId : currentTeam.getRiderIds()) {
80             removeRider(riderId);
81         }
82         allTeams.remove(teamIndex);
83     }
84 }
85 /**
86  * @return All the teamId's that are currently in the
        system as an int[]
87  */
88
89     int[] getTeams(){
90         int [] allTeamIds = new int[allTeams.size()];
91         for (int i=0; i<allTeams.size();i++){
92             allTeamIds[i]=allTeams.get(i).getId();
93         }
94         return allTeamIds;
95     }
96 /**
97  * @param teamId int - A teamId to get RidersId in that
        team. If the ID doesn't exist IDNotRecognisedException
        is thrown.
98  * @throws IDNotRecognisedException teamId int - If the ID
        doesn't exist IDNotRecognisedException is thrown.
99  * @return All the riderId's in a team as an int[]
100     */
101     int[] getTeamRiders(int teamId) throws
        IDNotRecognisedException{
102         Team currentTeam = getTeam(teamId);
103         return currentTeam.getRiderIds();
104     }
105 }
106 /**
107  * @return All team names in the system as an String[]
108  */
109     String[] getTeamsNames(){
110         String [] allTeamNames = new String[allTeams.size()];
111         for (int i=0; i<allTeams.size();i++){
112             allTeamNames[i] = allTeams.get(i).getTeamName();
113         }
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
128      *                      index. If the ID doesn't exist
129      *                      IDNotRecognisedException is thrown.
130      * @throws IDNotRecognisedException teamId int - If the ID
131      *                      doesn't exist IDNotRecognisedException is thrown.
132      * @return An int which is the index that maps to the
133      *                      teamId.
134      */
135     int getIndexForTeamId(int teamId) throws
136         IDNotRecognisedException{
137         int index = -1;
138         if (allTeams.size() == 0){
139             throw new IDNotRecognisedException("No Team exists
140             with that ID");
141         }
142         for (int i=0; i<allTeams.size();i++){
143             if (allTeams.get(i).getId()==teamId){
144                 index = i;
145                 break;
146             }
147         }
148         if (index == -1){
149             throw new IDNotRecognisedException("No rider exists
150             with that ID");
151         }
152         return index;
153     }
154     /**
155      * @param teamId int - A teamId of a team to search for its
156      *                      object. If the ID doesn't exist
157      *                      IDNotRecognisedException is thrown.
158      * @throws IDNotRecognisedException teamId int - If the ID
159      *                      doesn't exist IDNotRecognisedException is thrown.
160      * @return A Team object with the teamId parsed.
161      */
162     Team getTeam(int teamId) throws IDNotRecognisedException{
163         int teamIndex = getIndexForTeamId(teamId);
164         return allTeams.get(teamIndex);
165     }

```

```

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 }

```