1 CyclingPortal.java

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

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

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

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

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

```
196             return elapsedTime;
197         }
198 
199         @Override
200         public void deleteRiderResultsInStage(int stageId, int
                riderId) throws IDNotRecognisedException {
201             Stage.getStage(stageId);
202             riderManager.getRider(riderId);
203             // above should throw exceptions if IDs are not in
                    system
204             Result.removeResult(stageId, riderId);
205         }
206 
207         @Override
208         public int[] getRidersRankInStage(int stageId) throws
                IDNotRecognisedException {
209             Result[] results = Result.getResultsInStage(stageId);
210             int[] riderRanks = new int[results.length];
211             Arrays.fill(riderRanks, -1);
212             for(Result r : results) {
213                 for(int i=0; i<riderRanks.length; i++) {
214                     if(riderRanks[i] == -1) {
215                         riderRanks[i] = r.getRiderId();
216                         break;
217                     } else if(r.getTotalElasped().isBefore(Result.
                            getResult(stageId, riderRanks[i]).
                            getTotalElasped())) {
218                         int temp;
219                         int prev = r.getRiderId();
220                         for(int j=i; j<riderRanks.length; j++) {
221                             temp = riderRanks[j];
222                             riderRanks[j] = prev;
223                             prev = temp;
224                             if(prev == -1) {
225                                 break;
226                             }
227                         }
228                         break;
229                     }
230                 }
231             }
232             return riderRanks;
233         }
234 
235         @Override
236         public LocalTime[] getRankedAdjustedElapsedTimesInStage(int
                 stageId) throws IDNotRecognisedException {
237             int[] riderRanks = this.getRidersRankInStage(stageId);
238             LocalTime[] out = new LocalTime[riderRanks.length];
239             for(int i=0; i<out.length; i++) {
```

```
240             Result r = Result.getResult(stageId, riderRanks[i])
                    ;
241             LocalTime[] checkpoints = r.getCheckpoints();
242             LocalTime[] adjustedTimes = r.adjustedCheckpoints()
                    ;
243             out[i] = adjustedTimes[0];
244             LocalTime adjustedSplit;
245             for(int j=0; j<adjustedTimes.length; j++) {
246                 adjustedSplit = Result.getElapsed(adjustedTimes
                        [j], checkpoints[j]);
247                 System.out.println(adjustedSplit.toString());
248                 out[i] = out[i].plusHours(adjustedSplit.getHour
                        ());
249                 out[i] = out[i].plusMinutes(adjustedSplit.
                        getMinute());
250                 out[i] = out[i].plusSeconds(adjustedSplit.
                        getSecond());
251             }
252         }
253         return out;
254     }
255
256     @Override
257     public int[] getRidersPointsInStage(int stageId) throws
            IDNotRecognisedException {
258         StageType type = Stage.getStageType(stageId);
259         int[] points = new int[Result.getResultsInStage(stageId
                ).length];
260         int[] distribution = new int[15];
261         // distributions from https://en.wikipedia.org/wiki/
                Points_classification_in_the_Tour_de_France
262         switch(type) {
263             case FLAT:
264                 distribution = new int
                        []{50,30,20,18,16,14,12,10,8,7,6,5,4,3,2};
265                 break;
266             case MEDIUM_MOUNTAIN:
267                 distribution = new int
                        []{30,25,22,19,17,15,13,11,9,7,6,5,4,3,2};
268                 break;
269             case HIGH_MOUNTAIN:
270                 distribution = new int
                        []{20,17,15,13,11,10,9,8,7,6,5,4,3,2,1};
271                 break;
272             case TT:
273                 distribution = new int
                        []{20,17,15,13,11,10,9,8,7,6,5,4,3,2,1};
274                 break;
275         }
276         for(int i=0; i<Math.min(points.length, distribution.
```

```
            length); i++) {
277             points[i] = distribution[i];
278         }
279         return points;
280     }
281 
282     @Override
283     public int[] getRidersMountainPointsInStage(int stageId)
            throws IDNotRecognisedException {
284         Result[] results = Result.getResultsInStage(stageId);
285         // All results refering to the stage with id *stageId*
286         int[] riders = getRidersRankInStage(stageId);
287         // An int array of rider ids, from first to last
288         int[] segments = Stage.getSegments(stageId);
289         // An int array of the segment ids in the stage
290         int[] points = new int[riders.length];
291         // The int in position i is the number of points to be
                awarded to the rider with id riders[i]
292         for(int s=0; s<segments.length; s++) {
293             SegmentType type = Segment.getSegmentType(segments[
                    s]);
294             int[] distribution = new int[1];
295             // The points to be awarded in order for the
                    segment
296             switch(type) {
297                 case C4:
298                     distribution = new int[]{1};
299                     break;
300                 case C3:
301                     distribution = new int[]{2,1};
302                     break;
303                 case C2:
304                     distribution = new int[]{5,3,2,1};
305                     break;
306                 case C1:
307                     distribution = new int[]{10,8,6,4,2,1};
308                     break;
309                 case HC:
310                     distribution = new int
                            []{20,15,12,10,8,6,4,2};
311                     break;
312                 case SPRINT:
313             }
314             // get ranks for segment
315             int[] riderRanks = new int[results.length];
316             Arrays.fill(riderRanks, -1);
317             for(Result r : results) {
318                 for(int i=0; i<riderRanks.length; i++) {
319                     if(riderRanks[i] == -1) {
320                         riderRanks[i] = r.getRiderId();
```

```
321                         break;
322                     } else if(r.getCheckpoints()[s].isBefore(
                            Result.getResult(stageId, riderRanks[i])
                            .getCheckpoints()[s])) {
323                         int temp;
324                         int prev = r.getRiderId();
325                         for(int j=i; j<riderRanks.length; j++)
                                {
326                             temp = riderRanks[j];
327                             riderRanks[j] = prev;
328                             prev = temp;
329                             if(prev == -1) {
330                                 break;
331                             }
332                         }
333                         break;
334                     }
335                 }
336             }
337             ArrayList<Integer> ridersArray = new ArrayList<
                    Integer>();
338             for(int r : riders) { ridersArray.add(r); }
339             for(int i=0; i<Math.min(points.length, distribution
                    .length); i++) {
340                 int overallPos = ridersArray.indexOf(riderRanks
                        [i]);
341                 if(overallPos<points.length && overallPos!=-1)
                        {
342                     points[overallPos] += distribution[i];
343                 }
344             }
345         }
346         return points;
347     }
348
349     @Override
350     public void eraseCyclingPortal() {
351
352         Team.teamNames.clear();
353         Team.teamTopId = 0;
354         Rider.ridersTopId = 0;
355
356         RiderManager.allRiders.clear();
357         RiderManager.allTeams.clear();
358
359
360         Race.allRaces.clear();
361         Race.removedIds.clear();
362         Race.loadId();
363
```

```
364         Segment.allSegments.clear();
365         Segment.removedIds.clear();
366         Segment.loadId();
367 
368         Stage.allStages.clear();
369         Stage.removedIds.clear();
370         Stage.loadId();
371 
372         Result.allResults.clear();
373 
374 
375     }
376 
377     @Override
378     public void saveCyclingPortal(String filename) throws
            IOException {
379         try {
380             FileOutputStream fos = new FileOutputStream(
                    filename);
381             ObjectOutputStream oos = new ObjectOutputStream(fos
                    );
382             ArrayList<ArrayList> allObj = new ArrayList<>();
383             allObj.add(RiderManager.allTeams);
384             allObj.add(RiderManager.allRiders);
385             allObj.add(Stage.allStages);
386             allObj.add(Stage.removedIds);
387             allObj.add(Race.allRaces);
388             allObj.add(Race.removedIds);
389             allObj.add(Result.allResults);
390             allObj.add(Segment.allSegments);
391             allObj.add(Segment.removedIds);
392 
393             oos.writeObject(allObj);
394 
395             oos.flush();
396             oos.close();
397 
398         } catch (IOException ex) {
399             ex.printStackTrace();
400         }
401 
402     }
403 
404     @Override
405     public void loadCyclingPortal(String filename) throws
            IOException, ClassNotFoundException {
406         try {
407 
408             FileInputStream fis = new FileInputStream(filename)
                    ;
```

```
409             ObjectInputStream ois = new ObjectInputStream(fis);
410             ArrayList<Object> allObjects = new ArrayList<>();
411             ArrayList<Team> allTeams = new ArrayList<>();
412             ArrayList<Rider> allRiders = new ArrayList<>();
413             ArrayList<Result> allResults = new ArrayList<Result
                    >();
414             ArrayList<Race> allRaces = new ArrayList<Race>();
415             ArrayList<Stage> allStages = new ArrayList<Stage>()
                    ;
416             ArrayList<Segment> allSegments = new ArrayList<
                    Segment>();
417             ArrayList<Integer> removedIds = new ArrayList<>();
418
419             Class<?> classFlag = null;
420
421             allObjects = (ArrayList) ois.readObject();
422             for (Object tempObj : allObjects){
423                 ArrayList Objects = (ArrayList) tempObj;
424             for (Object obj : Objects){
425                 if (classFlag != null){
426                     if (obj.getClass() != classFlag && obj.
                            getClass() != Integer.class){
427                         if (classFlag == Race.class){
428                             Race.removedIds = removedIds;
429                         }
430                         if (classFlag == Segment.class){
431                             Segment.removedIds = removedIds;
432                         }
433                         if (classFlag == Stage.class){
434                             Stage.removedIds = removedIds;
435                         }
436                         classFlag = null;
437                         removedIds.clear();
438
439
440                     }
441                     else{
442                         Integer removedId = (Integer) obj;
443                         removedIds.add(removedId);
444
445                     }
446                 }
447                 String objClass = obj.getClass().getName();
448                 System.out.println(objClass);
449                 if (obj.getClass() == Rider.class){
450                     Rider newRider = (Rider) obj;
451                     allRiders.add(newRider);
452                     System.out.println("NEW RIDER");
453                 }
454                 if (obj.getClass() == Team.class){
```

```
455                     Team newTeam = (Team) obj;
456                     allTeams.add(newTeam);
457                     System.out.println("NEW TEAM");
458                 }
459                 if (obj.getClass() == Result.class){
460                     Result newResult = (Result) obj;
461                     allResults.add(newResult);
462                     System.out.println("NEW RESULT");
463                 }
464                 if (obj.getClass() == Stage.class){
465                     Stage newStage = (Stage) obj;
466                     allStages.add(newStage);
467                     System.out.println("NEW STAGE");
468                     classFlag = Stage.class;
469                 }
470                 if (obj.getClass() == Race.class){
471                     Race newRace = (Race) obj;
472                     allRaces.add(newRace);
473                     System.out.println("NEW Race");
474                     classFlag = Race.class;
475                 }
476                 if (obj.getClass() == Segment.class){
477                     Segment newSeg = (Segment) obj;
478                     allSegments.add(newSeg);
479                     System.out.println("NEW SEGMENT");
480                     classFlag = Segment.class;
481                 }
482 
483 
484                 System.out.println(obj.getClass());
485             }
486         }
487         if (classFlag == Race.class){
488             Race.removedIds = removedIds;
489         }
490         if (classFlag == Segment.class){
491             Segment.removedIds = removedIds;
492         }
493         if (classFlag == Stage.class){
494             Stage.removedIds = removedIds;
495         }
496 
497             this.riderManager.setAllTeams(allTeams);
498             this.riderManager.setAllRiders(allRiders);
499             Race.allRaces = allRaces;
500             Race.loadId();
501             Stage.allStages = allStages;
502             Stage.loadId();
503             Segment.allSegments = allSegments;
504             Segment.loadId();
```

```
505             Result.allResults = allResults;
506             ois.close();
507 
508         }
509         catch (Exception ex) {
510             ex.printStackTrace();
511         }
512 
513     }
514 
515     @Override
516     public void removeRaceByName(String name) throws
            NameNotRecognisedException {
517         boolean found = false;
518         for (int raceId : Race.getAllRaceIds()){
519             try {
520                 if (name == Race.getRaceName(raceId)){
521                     Race.removeRace(raceId);
522                 }
523             }
524             catch(Exception c){
525                 assert(false); // Exception will not throw by
                        for each condition
526                 // This try catch is easier than moving
                        exceptions to CyclingPortal level
527             }
528 
529         }
530         if (!found){ throw new NameNotRecognisedException("Name
                not in System.");}
531 
532     }
533 
534     @Override
535     public LocalTime[] getGeneralClassificationTimesInRace(int
            raceId) throws IDNotRecognisedException {
536         Race currentRace = Race.getRace(raceId);
537         int[] stageIds = currentRace.getStages();
538         int[] riderIds = this.riderManager.getRiderIds();
539         HashMap<Integer,Long> riderElaspedTime = new HashMap<
                Integer,Long>(); //Rider Id -> totalTime (long)
540         for (int riderId : riderIds){
541             riderElaspedTime.put(riderId,0L);
542         }
543         for (int stageId : stageIds){
544             Result[] temp = Result.getResultsInStage(stageId);
545             for(Result result: temp){
546                 int riderId = result.getRiderId();
547                 LocalTime getTotalElasped = result.
                        getTotalElasped();
```

```
548                 long timeTaken = getTotalElasped.toNanoOfDay();
549                 Long newTime = (Long)riderElaspedTime.get(
                        riderId)+timeTaken;
550                 riderElaspedTime.put(riderId,newTime);
551             }
552
553         }
554         long[][] riderTimePos = new long[riderIds.length][2];
555         int count = 0;
556         for (int riderId : riderIds){
557             Long finalRiderTime = riderElaspedTime.get(riderId)
                    ;// ## -> [[time,riderId],....] sort by time!
558             riderTimePos[count][0] = riderId;
559             riderTimePos[count][1] = finalRiderTime;
560             count++;
561         }
562         Arrays.sort(riderTimePos, Comparator.comparingDouble(o
                -> o[1]));
563         LocalTime[] finalTimes = new LocalTime[riderIds.length
                ];
564         count = 0;
565         for (long[] items : riderTimePos){
566             finalTimes[count]= LocalTime.ofNanoOfDay(items[1]);
567             count++;
568         }
569
570
571
572         return finalTimes;
573     }
574
575     @Override
576     public int[] getRidersPointsInRace(int raceId) throws
            IDNotRecognisedException {
577         ArrayList<Integer> order = new ArrayList<Integer>();
578         for(int riderId : getRidersGeneralClassificationRank(
                raceId)) {
579             order.add(riderId);
580         }
581         int[] out = new int[order.size()];
582         int[] stageRank, stagePoints;
583         for(int stageId : Race.getStages(raceId)) {
584             stageRank = getRidersRankInStage(stageId);
585             stagePoints = getRidersPointsInStage(stageId);
586             for(int i=0; i<stageRank.length; i++) {
587                 out[order.indexOf(stageRank[i])] += stagePoints
                        [i];
588             }
589         }
590         return out;
```

```
591     }
592
593     @Override
594     public int[] getRidersMountainPointsInRace(int raceId)
            throws IDNotRecognisedException {
595         ArrayList<Integer> order = new ArrayList<Integer>();
596         for(int riderId : getRidersGeneralClassificationRank(
                raceId)) {
597             order.add(riderId);
598         }
599         int[] out = new int[order.size()];
600         int[] stageRank, stagePoints;
601         for(int stageId : Race.getStages(raceId)) {
602             stageRank = getRidersRankInStage(stageId);
603             stagePoints = getRidersMountainPointsInStage(
                    stageId);
604             for(int i=0; i<stageRank.length; i++) {
605                 out[order.indexOf(stageRank[i])] += stagePoints
                        [i];
606             }
607         }
608         return out;
609     }
610
611     @Override
612     public int[] getRidersGeneralClassificationRank(int raceId)
             throws IDNotRecognisedException {
613         Race currentRace = Race.getRace(raceId);
614         int[] stageIds = currentRace.getStages();
615         int[] riderIds = this.riderManager.getRiderIds();
616         HashMap<Integer,Long> riderElaspedTime = new HashMap<
                Integer,Long>(); //Rider Id -> totalTime (long)
617         for (int riderId : riderIds){
618             riderElaspedTime.put(riderId,0L);
619         }
620         for (int stageId : stageIds){
621             Result[] temp = Result.getResultsInStage(stageId);
622             for(Result result: temp){
623                 int riderId = result.getRiderId();
624                 LocalTime getTotalElasped = result.
                        getTotalElasped();
625                 long timeTaken = getTotalElasped.toNanoOfDay();
626                 Long newTime = (Long)riderElaspedTime.get(
                        riderId)+timeTaken;
627                 riderElaspedTime.put(riderId,newTime);
628             }
629
630         }
631         long[][] riderTimePos = new long[riderIds.length][2];
632         int count = 0;
```

```
633             for (int riderId : riderIds){
634                 Long finalRiderTime = riderElaspedTime.get(riderId)
                        ;// ## -> [[time,riderId],....] sort by time!
635                 riderTimePos[count][0] = riderId;
636                 riderTimePos[count][1] = finalRiderTime;
637                 count++;
638             }
639             Arrays.sort(riderTimePos, Comparator.comparingDouble(o
                    -> o[1]));
640             int[] finalPos = new int[riderIds.length];
641             count = 0;
642             for (long[] items : riderTimePos){
643                 finalPos[count]= (int)items[0];
644                 count++;
645             }
646 
647             return finalPos;
648         }
649 
650         @Override
651         public int[] getRidersPointClassificationRank(int raceId)
                throws IDNotRecognisedException {
652             int[] order = getRidersGeneralClassificationRank(raceId
                    );
653             int[] points = getRidersPointsInRace(raceId);
654             int[] out = new int[order.length];
655             for(int i=0; i<out.length; i++) {
656                 int maxPoints = -1;
657                 int nextId = -1;
658                 for(int j=0; j<order.length; j++) {
659                     int id = order[j];
660                     if(id<0) { continue; }
661                     if(points[id] > maxPoints) {
662                         maxPoints = points[j];
663                         nextId = id;
664                     }
665                 }
666                 if(maxPoints < 0) {
667                     break;
668                 } else {
669                     out[i] = nextId;
670                     order[nextId] = -1;
671                 }
672             }
673             return out;
674         }
675 
676         @Override
677         public int[] getRidersMountainPointClassificationRank(int
                raceId) throws IDNotRecognisedException {
```

```
678         // effectively a clone of the method above
679         int[] order = getRidersGeneralClassificationRank(raceId
                );
680         int[] points = getRidersMountainPointsInRace(raceId);
681         int[] out = new int[order.length];
682         for(int i=0; i<out.length; i++) {
683             int maxPoints = -1;
684             int nextId = -1;
685             for(int j=0; j<order.length; j++) {
686                 int id = order[j];
687                 if(id<0) { continue; }
688                 if(points[id] > maxPoints) {
689                     maxPoints = points[j];
690                     nextId = id;
691                 }
692             }
693             if(maxPoints < 0) {
694                 break;
695             } else {
696                 out[i] = nextId;
697                 order[nextId] = -1;
698             }
699         }
700         return out;
701     }
702 }
```

2 Race.java

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

```
19      public static ArrayList<Race> allRaces = new ArrayList<Race
            >();
20
21      /**
22       * Loads the value of idMax.
23       */
24      public static void loadId(){
25          if(Race.allRaces.size()!=0) {
26              Race.idMax = Race.allRaces.get(Race.allRaces.size()
                    -1).getRaceId() + 1;
27          } else {
28              Race.idMax = 0;
29          }
30      }
31
32      /**
33       * @param raceId The ID of the race instance to fetch
34       * @return The race instance with the associated ID
35       * @throws IDNotRecognisedException If no race exists with
             the requested ID
36       */
37      public static Race getRace(int raceId) throws
            IDNotRecognisedException {
38          boolean removed = Race.removedIds.contains(raceId);
39          if(raceId<Race.idMax && raceId >= 0 && !removed) {
40              int index = raceId;
41              for(int j=0; j<Race.removedIds.size(); j++) {
42                  if(Race.removedIds.get(j) < raceId) {
43                      index--;
44                  }
45              }
46              return allRaces.get(index);
47          } else if (removed) {
48              throw new IDNotRecognisedException("no race
                    instance for raceID");
49          } else {
50              throw new IDNotRecognisedException("raceID out of
                    range");
51          }
52      }
53
54      /**
55       * @return An integer array of the race IDs of all races
56       */
57      public static int[] getAllRaceIds() {
58          int length = Race.allRaces.size();
59          int[] raceIdsArray = new int[length];
60          int i = 0;
61          for(Race race : allRaces) {
62              raceIdsArray[i] = race.getRaceId();
```

```
63              i++;
64          }
65          return raceIdsArray;
66      }
67
68      /**
69       * @param raceId The ID of the race instance to remove
70       * @throws IDNotRecognisedException If no race exists with
             the requested ID
71       */
72      public static void removeRace(int raceId) throws
            IDNotRecognisedException {
73          boolean removed = Race.removedIds.contains(raceId);
74          if(raceId<Race.idMax && raceId >= 0 && !removed) {
75              Race r = getRace(raceId);
76              for(int id : r.getStages()) {
77                  r.removeStageFromRace(id);
78              }
79              allRaces.remove(r);
80              removedIds.add(raceId);
81          } else if (removed) {
82              throw new IDNotRecognisedException("no race
                    instance for raceID");
83          } else {
84              throw new IDNotRecognisedException("raceID out of
                    range");
85          }
86      }
87
88      // Instance attributes
89      private int raceId;
90      private String raceName;
91      private String raceDescription;
92      private ArrayList<Integer> stageIds;
93
94      /**
95       * @param name String to be checked
96       * @return true if name is valid for the system
97       */
98      private static boolean validName(String name) {
99          if(name==null || name.equals("")) {
100             return false;
101         } else if(name.length()>30) {
102             return false;
103         } else if(name.contains(" ")) {
104             return false;
105         } else {
106             return true;
107         }
108     }
```

```
109
110     /**
111      * Race constructor; creates new race and adds to allRaces
             array.
112      *
113      * @param name The name of the new race
114      * @param description The description for the new race
115      * @throws IllegalNameException If name already exists in
             the system
116      * @throws InvalidNameException If name is empty/null,
             contains whitespace,
117      *                              or is longer than 30
             characters
118      */
119     public Race(String name, String description) throws
            IllegalNameException,
120                     InvalidNameException {
121         for(Race race : allRaces) {
122             if(race.getRaceName().equals(name)) {
123                 throw new IllegalNameException("name already
                        exists");
124             }
125         }
126         if(!validName(name)) {
127             throw new InvalidNameException("invalid name");
128         }
129         if(Race.removedIds.size() > 0) {
130             this.raceId = Race.removedIds.get(0);
131             Race.removedIds.remove(0);
132         } else {
133             this.raceId = idMax++;
134         }
135         this.raceName = name;
136         this.raceDescription = description;
137         this.stageIds = new ArrayList<Integer>();
138         Race.allRaces.add(this);
139     }
140
141     /**
142      * @return A string representation of the race instance
143      */
144     public String toString() {
145         String id = Integer.toString(this.raceId);
146         String name = this.raceName;
147         String description = this.raceDescription;
148         String list = this.stageIds.toString();
149         return String.format("Race[%s]: %s; %s; StageIds=%s;",
                id, name,
150                                 description, list);
151     }
```

```
152
153         /**
154          * @param id The ID of the race
155          * @return A string representation of the race instance
156          * @throws IDNotRecognisedException If no race exists with
                 the requested ID
157          */
158         public static String toString(int id) throws
                IDNotRecognisedException {
159             return getRace(id).toString();
160         }
161
162         /**
163          * @return The integer raceId for the race instance
164          */
165         public int getRaceId() { return this.raceId; }
166
167         /**
168          * @return The string raceName for the race instance
169          */
170         public String getRaceName() { return this.raceName; }
171
172         /**
173          * @param id The ID of the race
174          * @return The string raceName for the race with the
                 associated id
175          * @throws IDNotRecognisedException If no race exists with
                 the requested ID
176          */
177         public static String getRaceName(int id) throws
                IDNotRecognisedException {
178             return getRace(id).raceName;
179         }
180
181         /**
182          * @return The string raceDescription for the race instance
183          */
184         public String getRaceDescription() { return this.
                raceDescription; }
185
186         /**
187          * @param id The ID of the race
188          * @return The string raceDescription for the race with the
                 associated id
189          * @throws IDNotRecognisedException If no race exists with
                 the requested ID
190          */
191         public static String getRaceDescription(int id) throws
192                                                 IDNotRecognisedException
                                                        {
```

```
193             return getRace(id).raceDescription;
194         }
195 
196         /**
197          * @return An integer array of stage IDs for the race
                instance
198          */
199         public int[] getStages() {
200             int length = this.stageIds.size();
201             int[] stageIdsArray = new int[length];
202             for(int i=0; i<length; i++) {
203                 stageIdsArray[i] = this.stageIds.get(i);
204             }
205             return stageIdsArray;
206         }
207 
208         /**
209          * @param id The ID of the race
210          * @return An integer array of stage IDs for the race
                instance
211          * @throws IDNotRecognisedException If no race exists with
                the requested ID
212          */
213         public static int[] getStages(int id) throws
                IDNotRecognisedException {
214             Race race = getRace(id);
215             int length = race.stageIds.size();
216             int[] stageIdsArray = new int[length];
217             for(int i=0; i<length; i++) {
218                 stageIdsArray[i] = race.stageIds.get(i);
219             }
220             return stageIdsArray;
221         }
222 
223         /**
224          * @param name The new name for the race instance
225          */
226         public void setRaceName(String name) {
227             this.raceName = name;
228         }
229 
230         /**
231          * @param id The ID of the race to be updated
232          * @param name The new name for the race instance
233          * @throws IDNotRecognisedException If no race exists with
                the requested ID
234          */
235         public static void setRaceName(int id, String name) throws
236                                             IDNotRecognisedException {
237             getRace(id).setRaceName(name);
```

```
238     }
239
240     /**
241      * @param description The new description for the race
             instance
242      */
243     public void setRaceDescription(String description) {
244         this.raceDescription = description;
245     }
246
247     /**
248      * @param id The ID of the race to be updated
249      * @param description The new description for the race
             instance
250      * @throws IDNotRecognisedException If no race exists with
             the requested ID
251      */
252     public static void setRaceDescription(int id, String
         description) throws
253                                             IDNotRecognisedException
                                                 {
254         getRace(id).setRaceDescription(description);
255     }
256
257     /**
258      * Creates a new stage and adds the ID to the stageIds
             array.
259      *
260      * @param name The name of the new stage
261      * @param description The description of the new stage
262      * @param length The length of the new stage (in km)
263      * @param startTime The date and time at which the stage
             will be held
264      * @param type The StageType, used to determine the point
             distribution
265      * @return The ID of the new stage
266      */
267     public int addStageToRace(String name, String description,
         double length,
268                               LocalDateTime startTime,
                                      StageType type) throws
269                               IllegalNameException,
                                      InvalidNameException,
270                               InvalidLengthException {
271         Stage newStage = new Stage(name, description, length,
                startTime, type);
272         this.stageIds.add(newStage.getStageId());
273         return newStage.getStageId();
274     }
275
```

```
276     /**
277      * Creates a new stage and adds the ID to the stageIds
             array.
278      *
279      * @param id The ID of the race to which the stage will be
             added
280      * @param name The name of the new stage
281      * @param description The description of the new stage
282      * @param length The length of the new stage (in km)
283      * @param startTime The date and time at which the stage
             will be held
284      * @param type The StageType, used to determine the point
             distribution
285      * @return The ID of the new stage
286      * @throws IDNotRecognisedException If no race exists with
             the requested ID
287      */
288     public static int addStageToRace(int id, String name,
         String description,
289                                      double length,
                                             LocalDateTime startTime
                                             ,
290                                      StageType type) throws
291                                      IDNotRecognisedException,
292                                      IllegalNameException,
                                             InvalidNameException,
293                                      InvalidLengthException  {
294         return getRace(id).addStageToRace(name, description,
                length, startTime, type);
295     }
296
297     /**
298      * Removes a stageId from the array of stageIds for a race
             instance,
299      * as well as from the static array of all stages in the
             Stage class.
300      *
301      * @param stageId The ID of the stage to be removed
302      * @throws IDNotRecognisedException If no stage exists with
              the requested ID
303      */
304     private void removeStageFromRace(int stageId) throws
         IDNotRecognisedException {
305         if(this.stageIds.contains(stageId)) {
306             this.stageIds.remove(stageId);
307             Stage.removeStage(stageId);
308         } else {
309             throw new IDNotRecognisedException("stageID not
                    found in race");
310         }
```

```
311     }
312
313     /**
314      * Removes a stageId from the array of stageIds for a race
           instance,
315      * as well as from the static array of all stages in the
           Stage class.
316      *
317      * @param id The ID of the race to which the stage will be
           removed
318      * @param stageId The ID of the stage to be removed
319      * @throws IDNotRecognisedException If no stage exists with
            the requested ID
320      */
321     public static void removeStageFromRace(int id, int stageId)
          throws
322                                                IDNotRecognisedException
                                                       {
323         getRace(id).removeStageFromRace(stageId);
324     }
325
326     /**
327      * Removes a stageId from the array of stageIds for a race
           instance,
328      * as well as from the static array of all stages in the
           Stage class.
329      *
330      * @param stageId The ID of the stage to be removed
331      * @throws IDNotRecognisedException If no stage exists with
            the requested ID
332      */
333     public static void removeStage(int stageId) throws
          IDNotRecognisedException {
334         for(Race race : allRaces) {
335             if(race.stageIds.contains(stageId)) {
336                 race.removeStageFromRace(stageId);
337                 break;
338             }
339         }
340     }
341 }
```

3 Stage.java

```
1  package cycling;
2
3  import java.util.ArrayList;
4  import java.io.Serializable;
5  import java.time.LocalDateTime;
```

```
6  import java.time.format.DateTimeFormatter;
7
8  /**
9   * Stage encapsulates race stages, each of which has a number
        of associated
10  * Segments.
11  *
12  * @author Thomas Newbold
13  * @version 2.0
14  *
15  */
16 public class Stage implements Serializable {
17      // Static class attributes
18      private static int idMax = 0;
19      public static ArrayList<Integer> removedIds = new ArrayList
            <Integer>();
20      public static ArrayList<Stage> allStages = new ArrayList<
            Stage>();
21
22      /**
23       * Loads the value of idMax.
24       */
25      public static void loadId(){
26          if(Stage.allStages.size()!=0) {
27              Stage.idMax = Stage.allStages.get(Stage.allStages.
                    size()-1).getStageId() + 1;
28          } else {
29              Stage.idMax = 0;
30          }
31      }
32
33      /**
34       * @param stageId The ID of the stage instance to fetch
35       * @return The stage instance with the associated ID
36       * @throws IDNotRecognisedException If no stage exists with
              the requested ID
37       */
38      public static Stage getStage(int stageId) throws
            IDNotRecognisedException {
39          boolean removed = Stage.removedIds.contains(stageId);
40          if(stageId<Stage.idMax && stageId >= 0 && !removed) {
41              int index = stageId;
42              for(int j=0; j<Stage.removedIds.size(); j++) {
43                  if(Stage.removedIds.get(j) < stageId) {
44                      index--;
45                  }
46              }
47              return allStages.get(index);
48          } else if (removed) {
49              throw new IDNotRecognisedException("no stage
```

```
                instance for stageID");
50          } else {
51              throw new IDNotRecognisedException("stageId out of
                    range");
52          }
53      }
54
55      /**
56       * @return An integer array of the stage IDs of all stage
57       */
58      public static int[] getAllStageIds() {
59          int length = Stage.allStages.size();
60          int[] stageIdsArray = new int[length];
61          int i = 0;
62          for(Stage stage : allStages) {
63              stageIdsArray[i] = stage.getStageId();
64              i++;
65          }
66          return stageIdsArray;
67      }
68
69      /**
70       * @param stageId The ID of the stage instance to remove
71       * @throws IDNotRecognisedException If no stage exists with
               the requested ID
72       */
73      public static void removeStage(int stageId) throws
            IDNotRecognisedException {
74          boolean removed = Stage.removedIds.contains(stageId);
75          if(stageId<Stage.idMax && stageId >= 0 && !removed) {
76              Stage s = getStage(stageId);
77              for(int id : s.getSegments()) {
78                  s.removeSegmentFromStage(id);
79              }
80              allStages.remove(s);
81              removedIds.add(stageId);
82          } else if (removed) {
83              throw new IDNotRecognisedException("no stage
                    instance for stageID");
84          } else {
85              throw new IDNotRecognisedException("stageId out of
                    range");
86          }
87      }
88
89      // Instance attributes
90      private int stageId;
91      private StageState stageState;
92      private String stageName;
93      private String stageDescription;
```

```
94      private double stageLength;
95      private LocalDateTime stageStartTime;
96      private StageType stageType;
97      private ArrayList<Integer> segmentIds;
98
99      /**
100      * @param name String to be checked
101      * @return true if name is valid for the system
102      */
103     private static boolean validName(String name) {
104         if(name==null || name.equals("")) {
105             return false;
106         } else if(name.length()>30) {
107             return false;
108         } else if(name.contains(" ")) {
109             return false;
110         } else {
111             return true;
112         }
113     }
114
115     /**
116      * Stage constructor; creates a new stage and adds to
         allStages array.
117      *
118      * @param name The name of the new stage
119      * @param description The description of the new stage
120      * @param length The total length of the new stage
121      * @param startTime The start time for the new stage
122      * @param type The type of the new stage
123      * @throws IllegalNameException If name already exists in
         the system
124      * @throws InvalidNameException If name is empty/null,
         contains whitespace,
125      *                              or is longer than 30
         characters
126      * @throws InvalidLengthException If the length is less
         than 5km
127      */
128     public Stage(String name, String description, double length
         ,
129                  LocalDateTime startTime, StageType type)
                         throws
130                  IllegalNameException, InvalidNameException,
131                  InvalidLengthException {
132         for(Stage stage : allStages) {
133             if(stage.getStageName().equals(name)) {
134                 throw new IllegalNameException("name already
                        exists");
135             }
```

```
136             }
137             if(!validName(name)) {
138                 throw new InvalidNameException("invalid name");
139             }
140             if(length<5) {
141                 throw new InvalidLengthException("length less than
                        5km");
142             }
143             if(Stage.removedIds.size() > 0) {
144                 this.stageId = Stage.removedIds.get(0);
145                 Stage.removedIds.remove(0);
146             } else {
147                 this.stageId = idMax++;
148             }
149             this.stageState = StageState.BUILDING;
150             this.stageName = name;
151             this.stageDescription = description;
152             this.stageLength = length;
153             this.stageStartTime = startTime;
154             this.stageType = type;
155             this.segmentIds = new ArrayList<Integer>();
156             Stage.allStages.add(this);
157         }
158 
159         /**
160          * @return A string representation of the stage instance
161          */
162         public String toString() {
163             String id = Integer.toString(this.stageId);
164             String state;
165             switch (this.stageState) {
166                 case BUILDING:
167                     state = "In preperation";
168                     break;
169                 case WAITING:
170                     state = "Waiting for results";
171                     break;
172                 default:
173                     state = "null state";
174             }
175             String name = this.stageName;
176             String description = this.stageDescription;
177             String length = Double.toString(this.stageLength);
178             DateTimeFormatter formatter = DateTimeFormatter.
                    ofPattern("HH:hh dd-MM-yyyy");
179             String startTime = this.stageStartTime.format(formatter
                    );
180             String list = this.segmentIds.toString();
181             String type;
182             switch (this.stageType) {
```

```
183                 case FLAT:
184                     type = "Flat";
185                     break;
186                 case MEDIUM_MOUNTAIN:
187                     type = "Medium Mountain";
188                     break;
189                 case HIGH_MOUNTAIN:
190                     type = "High Mountain";
191                     break;
192                 case TT:
193                     type = "Time Trial";
194                     break;
195                 default:
196                     type = "null type";
197             }
198             return String.format("Stage[%s](%s): %s (%s); %s; %skm;
                    %s; SegmentIds=%s;",
199                                  id, state, name, type, description
                                      , length,
200                                  startTime, list);
201         }
202 
203         /**
204          * @param id The ID of the stage
205          * @return A string representation of the stage instance
206          * @throws IDNotRecognisedException If no stage exists with
                    the requested ID
207          */
208         public static String toString(int id) throws
                IDNotRecognisedException {
209             return getStage(id).toString();
210         }
211 
212         /**
213          * @return The integer stageId for the stage instance
214          */
215         public int getStageId() { return this.stageId; }
216 
217         /**
218          * @return The state of the stage instance
219          */
220         public StageState getStageState() { return this.stageState;
                }
221 
222         /**
223          * @param id The ID of the stage
224          * @return The state of the stage instance
225          * @throws IDNotRecognisedException If no stage exists with
                    the requested ID
226          */
```

```
227     public static StageState getStageState(int id) throws
228                                             IDNotRecognisedException
                                                    {
229         return getStage(id).getStageState();
230     }
231     /**
232      * @return The string raceName for the stage instance
233      */
234     public String getStageName() { return this.stageName; }
235
236     /**
237      * @param id The ID of the stage
238      * @return The string stageName for the stage with the
             associated id
239      * @throws IDNotRecognisedException If no stage exists with
              the requested ID
240      */
241     public static String getStageName(int id) throws
            IDNotRecognisedException {
242         return getStage(id).stageName;
243     }
244
245     /**
246      * @return The string stageDescription for the stage
             instance
247      */
248     public String getStageDescription() { return this.
            stageDescription; }
249
250     /**
251      * @param id The ID of the stage
252      * @return The string stageDescription for the stage with
             the associated id
253      * @throws IDNotRecognisedException If no stage exists with
              the requested ID
254      */
255     public static String getStageDescription(int id) throws
256                                               IDNotRecognisedException
                                                      {
257         return getStage(id).stageDescription;
258     }
259
260     /**
261      * @return The length of the stage instance
262      */
263     public double getStageLength() { return this.stageLength; }
264
265     /**
266      * @param id The ID of the stage
267      * @return The length of the stage instance
```

```
268         * @throws IDNotRecognisedException If no stage exists with
                the requested ID
269         */
270        public static double getStageLength(int id) throws
               IDNotRecognisedException {
271            return getStage(id).stageLength;
272        }
273
274        /**
275         * @return The start time for the stage instance
276         */
277        public LocalDateTime getStageStartTime() { return this.
               stageStartTime; }
278
279        /**
280         * @param id The ID of the stage
281         * @return The start time for the stage instance
282         * @throws IDNotRecognisedException If no stage exists with
                the requested ID
283         */
284        public static LocalDateTime getStageStartTime(int id)
               throws
285                                                          IDNotRecognisedException
                                                               {
286            return getStage(id).stageStartTime;
287        }
288
289        /**
290         * @return The type of the stage instance
291         */
292        public StageType getStageType() { return this.stageType; }
293
294        /**
295         * @param id The ID of the stage
296         * @return The type of the stage instance
297         * @throws IDNotRecognisedException If no stage exists with
                the requested ID
298         */
299        public static StageType getStageType(int id) throws
               IDNotRecognisedException {
300            return getStage(id).getStageType();
301        }
302
303        /**
304         * @return An integer array of segment IDs for the stage
                instance
305         */
306        public int[] getSegments() {
307            int length = this.segmentIds.size();
308            int[] segmentIdsArray = new int[length];
```

```
309             for(int i=0; i<length; i++) {
310                 segmentIdsArray[i] = this.segmentIds.get(i);
311             }
312             return segmentIdsArray;
313         }
314 
315         /**
316          * @param id The ID of the stage
317          * @return An integer array of segment IDs for the stage
                     instance
318          * @throws IDNotRecognisedException If no stage exists with
                     the requested ID
319          */
320         public static int[] getSegments(int id) throws
                IDNotRecognisedException {
321             Stage stage = getStage(id);
322             int length = stage.segmentIds.size();
323             int[] segmentIdsArray = new int[length];
324             for(int i=0; i<length; i++) {
325                 segmentIdsArray[i] = stage.segmentIds.get(i);
326             }
327             return segmentIdsArray;
328         }
329 
330         /**
331          * Updates the stage state from building to waiting for
                 results.
332          *
333          * @throws InvalidStageStateException If the stage is
                 already waiting for results
334          */
335         public void updateStageState() throws
                InvalidStageStateException {
336             if(this.stageState.equals(StageState.WAITING)) {
337                 throw new InvalidStageStateException("stage is
                        already waiting for results");
338             } else if(this.stageState.equals(StageState.BUILDING))
                    {
339                 this.stageState = StageState.WAITING;
340             }
341         }
342 
343         /**
344          * Updates the stage state from building to waiting for
                 results.
345          *
346          * @param id The ID of the stage to be updated
347          * @throws IDNotRecognisedException If no stage exists with
                     the requested ID
348          * @throws InvalidStageStateException If the stage is
```

```
            already waiting for results
349          */
350         public static void updateStageState(int id) throws
                IDNotRecognisedException,
351                                                 InvalidStageStateException
                                                        {
352             getStage(id).updateStageState();
353         }
354
355         /**
356          * @param name The new name for the stage instance
357          */
358         public void setStageName(String name) {
359             this.stageName = name;
360         }
361
362         /**
363          * @param id The ID of the stage to be updated
364          * @param name The new name for the stage instance
365          * @throws IDNotRecognisedException If no stage exists with
                  the requested ID
366          */
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
                instance
374          */
375         public void setStageDescription(String description) {
376             this.stageDescription = description;
377         }
378
379         /**
380          * @param id The ID of the stage to be updated
381          * @param description The new description for the stage
                instance
382          * @throws IDNotRecognisedException If no stage exists with
                  the requested ID
383          */
384         public static void setStageDescription(int id, String
                description) throws
385                                                    IDNotRecognisedException
                                                           {
386             getStage(id).setStageDescription(description);
387         }
388
389         /**
```

```
390         * @param length The new length for the stage instance
391         */
392        public void setStageLength(double length) {
393            this.stageLength = length;
394        }
395
396        /**
397         * @param id The ID of the stage to be updated
398         * @param length The new length for the stage instance
399         * @throws IDNotRecognisedException If no stage exists with
                the requested ID
400         */
401        public static void setStageLength(int id, double length)
               throws
402                                             IDNotRecognisedException
                                                   {
403            getStage(id).stageLength = length;
404        }
405
406        /**
407         * @param startTime The new start time for the stage
                instance
408         */
409        public void setStageStartTime(LocalDateTime startTime) {
410            this.stageStartTime = startTime;
411        }
412
413        /**
414         * @param id The ID of the stage to be updated
415         * @param startTime The new start time for the stage
                instance
416         * @throws IDNotRecognisedException If no stage exists with
                the requested ID
417         */
418        public static void setStageStartTime(int id, LocalDateTime
               startTime)
419                                                throws
                                                       IDNotRecognisedException
                                                        {
420            getStage(id).stageStartTime = startTime;
421        }
422
423        /**
424         * Creates a new stage and adds the ID to the stageIds
                array.
425         *
426         * @param location The location of the new segment
427         * @param type The type of the new segment
428         * @param averageGradient The average gradient of the new
                segment
```

```
429         * @param length The length (in km) of the new segment
430         * @throws InvalidLocationException If the segment finishes
                outside of the
431         *                                  bounds of the stage
432         * @throws InvalidStageStateException If the segment state
                is waiting for
433         *                                   results
434         * @throws InvalidStageTypeException If the stage type is a
                time-trial
435         *                                   (cannot contain
                segments)
436         */
437        public int addSegmentToStage(double location, SegmentType
               type,
438                                     double averageGradient, double
                                            length) throws
439                                     InvalidLocationException,
440                                     InvalidStageStateException,
441                                     InvalidStageTypeException {
442            if(location > this.getStageLength()) {
443                throw new InvalidLocationException("segment
                       finishes outside of stage bounds");
444            }
445            if(this.getStageState().equals(StageState.WAITING)) {
446                throw new InvalidStageStateException("stage is
                       waiting for results");
447            }
448            if(this.getStageType().equals(StageType.TT)) {
449                throw new InvalidStageTypeException("time trial
                       stages cannot contain segments");
450            }
451            Segment newSegment = new Segment(location, type,
                   averageGradient, length);
452            this.segmentIds.add(newSegment.getSegmentId());
453            return newSegment.getSegmentId();
454        }
455
456        /**
457         * Creates a new stage and adds the ID to the stageIds
                array.
458         *
459         * @param id The ID of the stage to which the segment will
                be added
460         * @param location The location of the new segment
461         * @param type The type of the new segment
462         * @param averageGradient The average gradient of the new
                segment
463         * @param length The length (in km) of the new segment
464         * @throws IDNotRecognisedException If no stage exists with
                the requested ID
```

```
465         * @throws InvalidLocationException If the segment finishes
                outside of the
466         *                                  bounds of the stage
467         * @throws InvalidStageStateException If the segment state
                is waiting for
468         *                                   results
469         * @throws InvalidStageTypeException If the stage type is a
                time-trial
470         *                                   (cannot contain
                segments)
471         */
472        public static int addSegmentToStage(int id, double location
               , SegmentType type,
473                                            double averageGradient,
                                                    double length)
                                                   throws
474                                            IDNotRecognisedException
                                                   ,
475                                            InvalidLocationException
                                                   ,
476                                            InvalidStageStateException
                                                   ,
477                                            InvalidStageTypeException
                                                    {
478            return getStage(id).addSegmentToStage(location, type,
                   averageGradient, length);
479        }
480
481        /**
482         * Removes a segmentId from the array of segmentIds for a
                stage instance,
483         * as well as from the static array of all segments in the
                Segment class.
484         *
485         * @param segmentId The ID of the segment to be removed
486         * @throws IDNotRecognisedException If no segment exists
                with the requested
487         *                                  ID
488         */
489        private void removeSegmentFromStage(int segmentId) throws
490                                            IDNotRecognisedException
                                                    {
491            if(this.segmentIds.contains(segmentId)) {
492                this.segmentIds.remove(segmentId);
493                Segment.removeSegment(segmentId);
494            } else {
495                throw new IDNotRecognisedException("segmentID not
                       found in race");
496            }
497        }
```

```
498
499     /**
500      * Removes a segmentId from the array of segmentIds for a
             stage instance,
501      * as well as from the static array of all segments in the
             Segment class.
502      *
503      * @param id The ID of the stage to which the segment will
             be removed
504      * @param segmentId The ID of the segment to be removed
505      * @throws IDNotRecognisedException If no segment exists
             with the requested
506      *                                  ID
507      */
508     public static void removeSegmentFromStage(int id, int
         segmentId) throws
509                                               IDNotRecognisedException
                                                      {
510         getStage(id).removeSegmentFromStage(segmentId);
511     }
512
513     /**
514      * Removes a segmentId from the array of segmentIds for a
             stage instance,
515      * as well as from the static array of all segments in the
             Segment class.
516      *
517      * @param segmentId The ID of the segment to be removed
518      * @throws IDNotRecognisedException If no segment exists
             with the requested
519      *                                  ID
520      */
521     public static void removeSegment(int segmentId) throws
         IDNotRecognisedException {
522         for(Stage stage : allStages) {
523             if(stage.segmentIds.contains(segmentId)) {
524                 stage.removeSegmentFromStage(segmentId);
525                 break;
526             }
527         }
528     }
529 }
```

4 StageState.java

```
1  package cycling;
2
3  /**
4   * This enum is used to represent the state of a stage.
```

```
 *
 * @author Thomas Newbold
 * @version 1.0
 *
 */
public enum StageState {

    /**
     * Used for stages still in preperation - i.e. segments are
         still being
     * added.
     */
    BUILDING,

    /**
     * Used for stages waiting for results
     */
    WAITING;
}
```

5 Segment.java

```
package cycling;

import java.io.Serializable;
import java.util.ArrayList;

/**
 * Segment encapsulates race segments
 *
 * @author Thomas Newbold
 * @version 2.0
 *
 */
public class Segment implements Serializable {
    // Static class attributes
    private static int idMax = 0;
    public static ArrayList<Integer> removedIds = new ArrayList
        <Integer>();
    public static ArrayList<Segment> allSegments = new
        ArrayList<Segment>();

    /**
     * Loads the value of idMax.
     */
    public static void loadId(){
        if(Segment.allSegments.size()!=0) {
            Segment.idMax = Segment.allSegments.get(-1).
                getSegmentId() + 1;
```

```
25          } else {
26              Segment.idMax = 0;
27          }
28      }
29
30      /**
31       * @param segmentId The ID of the segment instance to fetch
32       * @return The segment instance with the associated ID
33       * @throws IDNotRecognisedException If no segment exists
             with the requested
34       *                                     ID
35       */
36      public static Segment getSegment(int segmentId) throws
37                                       IDNotRecognisedException {
38          boolean removed = Segment.removedIds.contains(segmentId
                );
39          if(segmentId<Segment.idMax && segmentId >= 0 && !
                removed) {
40              int index = segmentId;
41              for(int j=0; j<Segment.removedIds.size(); j++) {
42                  if(Segment.removedIds.get(j) < segmentId) {
43                      index--;
44                  }
45              }
46              return allSegments.get(index);
47          } else if (removed) {
48              throw new IDNotRecognisedException("no segment
                    instance for "+
49                                                 "segmentId");
50          } else {
51              throw new IDNotRecognisedException("segmentId out
                    of range");
52          }
53      }
54
55      /**
56       * @return An integer array of the segment IDs of all
             segment
57       */
58      public static int[] getAllSegmentIds() {
59          int length = Segment.allSegments.size();
60          int[] segmentIdsArray = new int[length];
61          int i = 0;
62          for(Segment segment : allSegments) {
63              segmentIdsArray[i] = segment.getSegmentId();
64              i++;
65          }
66          return segmentIdsArray;
67      }
68
```

```
69      /**
70       * @param segmentId The ID of the segment instance to
             remove
71       * @throws IDNotRecognisedException If no segment exists
             with the requested
72       *                                     ID
73       */
74      public static void removeSegment(int segmentId) throws
75                                       IDNotRecognisedException {
76          boolean removed = Segment.removedIds.contains(segmentId
                );
77          if(segmentId<Segment.idMax && segmentId >= 0 && !
                removed) {
78              Segment s = getSegment(segmentId);
79              allSegments.remove(s);
80              removedIds.add(segmentId);
81          } else if (removed) {
82              throw new IDNotRecognisedException("no segment
                    instance for "+
83                                                 "segmentId");
84          } else {
85              throw new IDNotRecognisedException("segmentId out
                    of range");
86          }
87      }
88
89      // Instance attributes
90      private int segmentId;
91      private double segmentLocation;
92      private SegmentType segmentType;
93      private double segmentAverageGradient;
94      private double segmentLength;
95
96      /**
97       * Segment constructor; creates a new segment and adds to
             allSegment array.
98       *
99       * @param location The location of the finish of the new
             segment in the stage
100      * @param type The type of the new segment
101      * @param averageGradient The average gradient of the new
             segment
102      * @param length The length of the new segment
103      */
104     public Segment(double location, SegmentType type, double
            averageGradient,
105                    double length) {
106         if(Segment.removedIds.size() > 0) {
107             this.segmentId = Segment.removedIds.get(0);
108             Segment.removedIds.remove(0);
```

```
109             } else {
110                 this.segmentId = idMax++;
111             }
112             this.segmentLocation = location;
113             this.segmentType = type;
114             this.segmentAverageGradient = averageGradient;
115             this.segmentLength = length;
116             Segment.allSegments.add(this);
117         }
118 
119         /**
120          * @return A string representation of the segment instance
121          */
122         public String toString() {
123             String id = Integer.toString(this.segmentId);
124             String location = Double.toString(this.segmentLocation)
                    ;
125             String type;
126             switch (this.segmentType) {
127                 case SPRINT:
128                     type = "Sprint";
129                     break;
130                 case C4:
131                     type = "Category 4 Climb";
132                     break;
133                 case C3:
134                     type = "Category 3 Climb";
135                     break;
136                 case C2:
137                     type = "Category 2 Climb";
138                     break;
139                 case C1:
140                     type = "Category 1 Climb";
141                     break;
142                 case HC:
143                     type = "Hors Categorie";
144                     break;
145                 default:
146                     type = "null category";
147             }
148             String averageGrad = Double.toString(this.
                    segmentAverageGradient);
149             String length = Double.toString(this.segmentLength);
150             return String.format("Segment[%s]: %s; %skm; Location=%
                    s; Gradient=%s;",
151                                  id, type, length, location,
                                         averageGrad);
152         }
153 
154         /**
```

```
155         * @param id The ID of the segment
156         * @return A string representation of the segment instance
157         * @throws IDNotRecognisedException If no segment exists
                with the requested
158         *                                    ID
159         */
160        public static String toString(int id) throws
               IDNotRecognisedException {
161            return getSegment(id).toString();
162        }
163
164        /**
165         * @return The integer segmentId for the segment instance
166         */
167        public int getSegmentId() { return this.segmentId; }
168
169        /**
170         * @return The integer representing the location of the
               segment instance
171         */
172        public double getSegmentLocation() { return this.
               segmentLocation; }
173
174        /**
175         * @param id The ID of the segment
176         * @return The integer representing the location of the
               segment instance
177         * @throws IDNotRecognisedException If no segment exists
               with the requested
178         *                                    ID
179         */
180        public static double getSegmentLocation(int id) throws
181                                                  IDNotRecognisedException
                                                      {
182            return getSegment(id).segmentLocation;
183        }
184
185        /**
186         * @return The type of the segment instance
187         */
188        public SegmentType getSegmentType() { return this.
               segmentType; }
189
190        /**
191         * @param id The ID of the segment
192         * @return The type of the segment instance
193         * @throws IDNotRecognisedException If no segment exists
               with the requested
194         *                                    ID
195         */
```

```
196     public static SegmentType getSegmentType(int id) throws
197                                                 IDNotRecognisedException
                                                        {
198         return getSegment(id).segmentType;
199     }
200
201     /**
202      * @return The average gradient of the segment instance
203      */
204     public double getSegmentAverageGradient() {
205         return this.segmentAverageGradient;
206     }
207
208     /**
209      * @param id The ID of the segment
210      * @return The average gradient of the segment instance
211      * @throws IDNotRecognisedException If no segment exists
             with the requested
212      *                                  ID
213      */
214     public static double getSegmentAverageGradient(int id)
        throws
215                                                       IDNotRecognisedException
                                                            {
216         return getSegment(id).segmentAverageGradient;
217     }
218
219     /**
220      * @return The length of the segment instance
221      */
222     public double getSegmentLength() { return this.
        segmentLength; }
223
224     /**
225      * @param id The ID of the segment
226      * @return The length of the segment instance
227      * @throws IDNotRecognisedException If no segment exists
             with the requested
228      *                                  ID
229      */
230     public static double getSegmentLength(int id) throws
        IDNotRecognisedException {
231         return getSegment(id).segmentLength;
232     }
233
234     /**
235      * @param location The new location for the segment
             instance
236      */
237     public void setSegmentLocation(double location) {
```

```
238             this.segmentLocation = location;
239         }
240 
241         /**
242          * @param id The ID of the segment to be updated
243          * @param location The new location for the segment
                 instance
244          * @throws IDNotRecognisedException If no segment exists
                 with the requested
245          *                                     ID
246          */
247         public static void setSegmentLocation(int id, double
                location) throws
248                                                 IDNotRecognisedException
                                                        {
249             getSegment(id).setSegmentLocation(location);
250         }
251 
252         /**
253          * @param type The new type for the segment instance
254          */
255         public void setSegmentType(SegmentType type) {
256             this.segmentType = type;
257         }
258 
259         /**
260          * @param id The ID of the segment to be updated
261          * @param type The new type for the segment instance
262          * @throws IDNotRecognisedException If no segment exists
                 with the requested
263          *                                     ID
264          */
265         public static void setSegmentType(int id, SegmentType type)
                 throws
266                                           IDNotRecognisedException
                                                  {
267             getSegment(id).setSegmentType(type);
268         }
269 
270         /**
271          * @param averageGradient The new average gradient for the
                 segment instance
272          */
273         public void setSegmentAverageGradient(double
                averageGradient) {
274             this.segmentAverageGradient = averageGradient;
275         }
276 
277         /**
278          * @param id The ID of the segment to be updated
```

```
279         * @param averageGradient The new average gradient for the
                segment instance
280         * @throws IDNotRecognisedException If no segment exists
                with the requested
281         *                                     ID
282         */
283        public static void setSegmentAverageGradient(int id, double
                averageGradient)
284                                                          throws
                                                              IDNotRecognisedException
                                                               {
285            getSegment(id).setSegmentAverageGradient(
                   averageGradient);
286        }
287
288        /**
289         * @param length The new length for the segment instance
290         */
291        public void setSegmentLength(double length) {
292            this.segmentLength = length;
293        }
294
295        /**
296         * @param id The ID of the segment to be updated
297         * @param length The new length for the segment instance
298         * @throws IDNotRecognisedException If no segment exists
                with the requested
299         *                                     ID
300         */
301        public static void setSegmentLength(int id, double length)
               throws
302                                               IDNotRecognisedException
                                                    {
303            getSegment(id).setSegmentLength(length);
304        }
305    }
```

6 Result.java

```
1   package cycling;
2
3   import java.util.ArrayList;
4   import java.util.Arrays;
5   import java.io.Serializable;
6   import java.time.LocalTime;
7   import java.time.format.DateTimeFormatter;
8   import java.time.temporal.ChronoUnit;
9
10  /**
```

```
11   * Result encapsulates rider results per stage, and handles
         time adjustments and
12   * rankings (scoring is done externally based on points
         distributions defined in
13   * Cycling Portal)
14   *
15   * @author Thomas Newbold
16   * @version 1.1
17   */
18  public class Result implements Serializable {
19      // Static class attributes
20      public static ArrayList<Result> allResults = new ArrayList<
            Result>();
21
22      /**
23       * @param stageId The ID of the stage
24       * @return An array of all results for a stage
25       */
26      public static Result[] getResultsInStage(int stageId) {
27          ArrayList<Result> stage = new ArrayList<Result>();
28          for(Result r : allResults) {
29              stage.add(r);
30          }
31          stage.removeIf(r -> r.getStageId()!=stageId);
32          Result[] resultsForStage = new Result[stage.size()];
33          for(int i=0; i<stage.size(); i++) {
34              resultsForStage[i] = stage.get(i);
35          }
36          return resultsForStage;
37      }
38
39      /**
40       * @param riderId The ID of the driver
41       * @return An array of all results for a driver
42       */
43      public static Result[] getResultsForRider(int riderId) {
44          ArrayList<Result> rider = new ArrayList<Result>(
                allResults);
45          rider.removeIf(r -> r.getRiderId()!=riderId);
46          Result[] resultsForRider = new Result[rider.size()];
47          for(int i=0; i<rider.size(); i++) {
48              resultsForRider[i] = rider.get(i);
49          }
50          return resultsForRider;
51      }
52
53      // Instance attributes
54      private int stageId;
55      private int riderId;
56      private LocalTime[] checkpoints;
```

```
57
58      /**
59       * Result constructor; creates a new result entry and adds
             to the
60       * allResults array.
61       *
62       * @param sId The ID of the stage the result refers to
63       * @param rId The ID of the rider who achieved the result
64       * @param check An array of times at which the rider
             reached each
65       *              checkpoint (including start and finish)
66       */
67      public Result(int sId, int rId, LocalTime... check) {
68          this.stageId = sId;
69          this.riderId = rId;
70          this.checkpoints = check;
71          Result.allResults.add(this);
72      }
73
74      /**
75       * @return A string representation of the Result instance
76       */
77      public String toString() {
78          String sId = Integer.toString(this.stageId);
79          String rId = Integer.toString(this.riderId);
80          int l = this.getCheckpoints().length;
81          String times[] = new String[l];
82          DateTimeFormatter formatter = DateTimeFormatter.
                ofPattern("HH:mm:ss");
83          for(int i=0; i<l; i++) {
84              times[i] = this.getCheckpoints()[i].format(
                    formatter);
85          }
86          return String.format("Stage[%s]-Rider[%s]: SplitTimes=%
                s; Total=%s",
87                               sId, rId, Arrays.toString(times),
88                               getTotalElasped().format(formatter
                                    ));
89      }
90
91      /**
92       * @param sId The ID of the stage of the result instance
93       * @param rId The ID of the associated rider to the result
             instance
94       * @return The Result instance
95       * @throws IDNotRecognisedException If an instance for the
             rider/stage
96       *                                  combination is not
             found in the
97       *                                  allResults array
```

```
98          */
99          public static Result getResult(int sId, int rId) throws
                IDNotRecognisedException {
100             for(Result r : allResults) {
101                 if(r.getRiderId()==rId && r.getStageId()==sId) {
102                     return r;
103                 }
104             }
105             throw new IDNotRecognisedException("results not found
                    for rider in stage");
106         }
107
108         /**
109          * @param sId The ID of the stage of the result instance to
                 remove
110          * @param rId The ID of the associated rider to the result
                 instance to remove
111          * @throws IDNotRecognisedException If an instance for the
                 rider/stage
112          *                                  combination is not
                 found in the
113          *                                  allResults array
114          */
115         public static void removeResult(int sId, int rId) throws
                IDNotRecognisedException {
116             for(Result r : allResults) {
117                 if(r.getRiderId()==rId && r.getStageId()==sId) {
118                     allResults.remove(r);
119                     break;
120                 }
121             }
122             throw new IDNotRecognisedException("results not found
                    for rider in stage");
123         }
124
125         /**
126          * @return The stageId of the stage the result refers to
127          */
128         public int getStageId() { return this.stageId; }
129
130         /**
131          * @return The riderId of the rider associated with the
                 result
132          */
133         public int getRiderId() { return this.riderId; }
134
135         /**
136          * @return An array of the split times between each
                 checkpoint
137          */
```

```
138     public LocalTime[] getCheckpoints() {
139         LocalTime[] out = new LocalTime[this.checkpoints.length
                -1];
140         for(int n=0;n<this.checkpoints.length-1; n++) {
141             out[n] = getElapsed(checkpoints[n],checkpoints[n
                    +1]);
142         }
143         return out;
144     }
145 
146     /**
147      * @return The total time elapsed between the start and end
               checkpoints
148      */
149     public LocalTime getTotalElasped() {
150         LocalTime[] times = this.checkpoints;
151         return Result.getElapsed(times[0], times[times.length
                -1]);
152     }
153 
154     /**
155      * @param a Start time
156      * @param b End time
157      * @return The time difference between two times, a and b
158      */
159     public static LocalTime getElapsed(LocalTime a, LocalTime b
            ) {
160         int hours = (int)a.until(b, ChronoUnit.HOURS);
161         int minuites = (int)a.until(b, ChronoUnit.MINUTES);
162         int seconds = (int)a.until(b, ChronoUnit.SECONDS);
163         return LocalTime.of(hours%24, minuites%60, seconds%60);
164     }
165 
166     /**
167      * @return An array of the checkpoint times, adjusted to a
               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         * @param n The index of the checkpoint to adjust
182         * @return The adjusted time for checkpoint n
183         */
184        public LocalTime adjustedCheckpoint(int n) {
185            for(int i=0; i<allResults.size(); i++) {
186                Result r = allResults.get(i);
187                if(r.getRiderId()==this.getRiderId() && r.
                       getStageId()==this.getStageId()) {
188                    continue;
189                }
190                LocalTime selfTime = this.getCheckpoints()[n];
191                LocalTime rTime = r.getCheckpoints()[n];
192                if(selfTime.until(rTime, ChronoUnit.SECONDS)<1) {
193                    return r.adjustedCheckpoint(n);
194                } else {
195                    return selfTime;
196                }
197            }
198            return null;
199        }
200    }
```

7 Team.java

```
1   package cycling;
2   import java.io.Serializable;
3   import java.util.ArrayList;
4   /**
5    * Team Class holds the teamId,name,description and riderIds
         belonging to that team.
6    *
7    *
8    * @author Ethan Ray
9    * @version 1.0
10   *
11   */
12
13  public class Team implements Serializable {
14      public static ArrayList<String> teamNames = new ArrayList
            <>();
15      public static int teamTopId = 0;
16
17      private int teamID;
18      private String name;
19      private String description;
20      private ArrayList<Integer> riderIds = new ArrayList<>();
21
22
23      /**
```

```
24          * @param name String - A name for the team, , If the name
                is null, empty, has more than 30 characters, or has
                white spaces will throw InvaildNameException.
25          * @param description String - A description for the team.
26          * @throws IllegalNameException name String - Is a
                duplicate name of any other Team, IllegalNameException
                will be thrown.
27          * @throws InvailNameException name String - If the name is
                 null, empty, has more than 30 characters, or has white
                 spaces will throw InvaildNameException.
28          */
29         public Team(String name, String description) throws
               IllegalNameException, InvalidNameException
30         {
31             if (name == "" || name.length()>30 || name.contains(" "
                   )){
32                 throw new InvalidNameException("Team name cannot be
                       empty, longer than 30 characters , or has white
                       spaces.");
33             }
34             for (int i = 0;i<teamNames.size();i++){
35                 if (teamNames.get(i) == name){
36                     throw new IllegalNameException("That team name
                           already exsists!");
37                 }
38             }
39
40             teamNames.add(name);
41             this.teamID = teamTopId++;
42             this.name = name;
43             this.description = description;
44         }
45         /**
46          * @param rider Rider - A rider to add to the team.
47          */
48         public void addRider(Rider rider){
49
50             this.riderIds.add(rider.getRiderId());
51         }
52         /**
53          * @param riderId int - A riderId to be removed from the
                team.
54          */
55         public void removeRiderId(int riderId){
56             for (int i =0;i<this.riderIds.size();i++){
57                 if (this.riderIds.get(i)==riderId){
58                     this.riderIds.remove(i);
59                     break;
60                 }
61             }
```

```
62      }
63      /**
64       * @return An Array of integers - which are the riderIds in
             that team.
65       */
66      public int[] getRiderIds(){
67          int [] currentRiderIds = new int[this.riderIds.size()];
68          for (int i=0; i<this.riderIds.size();i++){
69              currentRiderIds[i]=this.riderIds.get(i);
70          }
71          return currentRiderIds;
72      }
73      /**
74       * @return A Integer - teamId of the team.
75       */
76      public int getId(){
77          return this.teamID;
78      }
79      /**
80       * @return A String - Name of the team.
81       */
82      public String getTeamName(){
83          return this.name;
84      }
85      /**
86       * @return A String - The description of the team.
87       */
88      public String getDescription(){
89          return this.description;
90      }
91  }
```

8 Rider.java

```
1   package cycling;
2
3   import java.io.Serializable;
4
5   /**
6    * Rider Class holds the riders teamId,riderId,name and
         yearOfBirth
7    *
8    *
9    * @author Ethan Ray
10   * @version 1.0
11   *
12   */
13
14
```

```java
public class Rider implements Serializable {
    public static int ridersTopId;
    private int riderId;
    private int teamID;
    private String name;
    private int yearOfBirth;


    /**
     * @param teamID int - A team Id that the rider will belong too
     * @param name String - A name for the rider, Has to be non-null or IllegalArgumentException is thrown.
     * @param yearOfBirth int - A year that the rider was born in. Has to be above 1900 or IllegalArgumentException is thrown.
     * @throws IllegalArgumentException name String - Has to be non-null or IllegalArgumentException is thrown.
     * @throws IllegalArgumentException yearOfBirth int - A year that the rider was born in. Has to be above 1900 or IllegalArgumentException is thrown.
     */
    public Rider(int teamID, String name, int yearOfBirth) throws IllegalArgumentException
    {
        this.riderId = ridersTopId++;
        this.teamID = teamID;
        if (name == "" || name == null){
            throw new IllegalArgumentException("Illegal name entered for rider");
        }
        this.name = name;
        if (yearOfBirth < 1900){
            throw new IllegalArgumentException("Illegal value for yearOfBirth given please enter a value above 1900.");
        }
        this.yearOfBirth = yearOfBirth;
    }
    /**
     * @return The RiderId of the rider.
     */
    public int getRiderId(){
        return this.riderId;
    }
    /**
     * @return The team Id that the rider belongs to/
     */
    public int getRiderTeamId(){
        return this.teamID;
```

```
54      }
55      /**
56       * @return The rider's name.
57       */
58      public String getRiderName(){
59          return this.name;
60      }
61      /**
62       * @return The the year of birth of the rider.
63       */
64      public int getRiderYOB(){
65          return this.yearOfBirth;
66      }
67
68  }
```

9 RiderManager.java

```
1   package cycling;
2
3   import java.io.Serializable;
4   import java.util.ArrayList;
5
6   public class RiderManager implements Serializable{
7       public static ArrayList<Rider> allRiders = new ArrayList
            <>();
8       public static ArrayList<Team> allTeams = new ArrayList<>();
9
10
11      /**
12       * @param teamID int - A team Id that the rider will belong
               too. If the ID doesn't exist IDNotRecognisedException
              is thrown.
13       * @param name String - A name for the rider, Has to be non
              -null or IllegalArgumentException is thrown.
14       * @param yearOfBirth int - A year that the rider was born
              in. Has to be above 1900 or IllegalArgumentException is
               thrown.
15       * @return riderId of the rider created.
16       * @throws IDNotRecognisedException teamId int - If the ID
              doesn't exist IDNotRecognisedException is thrown.
17       * @throws IllegalArgumentException yearOfBirth int - A
              year that the rider was born in. Has to be above 1900
              or IllegalArgumentException is thrown.
18       */
19      int createRider(int teamID, String name, int yearOfBirth)
            throws IDNotRecognisedException,IllegalArgumentException
            {
20          int teamIndex = getIndexForTeamId(teamID);
```

```
21          Rider newRider = new Rider(teamID,name,yearOfBirth);
22          allRiders.add(newRider);
23          Team ridersTeam = allTeams.get(teamIndex);
24          ridersTeam.addRider(newRider);
25          return newRider.getRiderId();
26      }
27      /**
28       * @param riderId int - A riderId of a rider to be removed.
             If the ID doesn't exist IDNotRecognisedException is
            thrown.
29       * @throws IDNotRecognisedException riderId int - If the ID
             doesn't exist IDNotRecognisedException is thrown.
30       */
31      void removeRider(int riderId) throws
            IDNotRecognisedException
32      {
33          int riderIndex = getIndexForRiderId(riderId);
34          int teamId = allRiders.get(riderIndex).getRiderTeamId()
                ;
35          int teamIndex = getIndexForTeamId(teamId);
36          Team riderTeam = allTeams.get(teamIndex);
37          riderTeam.removeRiderId(riderId);
38          allRiders.remove(riderIndex);
39      }
40      /**
41       * @param riderId int - A riderId of a rider to be searced
            for. If the ID doesn't exist IDNotRecognisedException
            is thrown.
42       * @throws IDNotRecognisedException riderId int - If the ID
             doesn't exist IDNotRecognisedException is thrown.
43       * @return An int which is the index that maps to the
            riderId.
44       */
45      int getIndexForRiderId(int riderId) throws
            IDNotRecognisedException{
46          int index =-1;
47          if (allRiders.size() == 0){
48              throw new IDNotRecognisedException("No rider exists
                    with that ID");
49          }
50          for (int i=0; i<allRiders.size();i++){
51              if (allRiders.get(i).getRiderId()==riderId){
52                  index = i;
53                  break;
54              }
55          }
56          if (index == -1){
57              throw new IDNotRecognisedException("No rider exists
                    with that ID");
58          }
```

```
59          return index;
60      }
61      /**
62       * @param name String - A name for the team, , If the name
             is null, empty, has more than 30 characters, or has
             white spaces will throw InvaildNameException.
63       * @param description String - A description for the team.
64       * @throws IllegalNameException name String - Is a
             duplicate name of any other Team, IllegalNameException
             will be thrown.
65       * @throws InvailNameException name String - If the name is
              null, empty, has more than 30 characters, or has white
             spaces will throw InvaildNameException.
66       */
67      int createTeam(String name, String description) throws
            IllegalNameException, InvalidNameException{
68          Team newTeam = new Team(name,description);
69          allTeams.add(newTeam);
70          return newTeam.getId();
71      }
72      /**
73       * @param teamId int - A teamId of a rider to be removed.
             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
                index. If the ID doesn't exist
                IDNotRecognisedException is thrown.
128         * @throws IDNotRecognisedException teamId int - If the ID
                doesn't exist IDNotRecognisedException is thrown.
129         * @return An int which is the index that maps to the
                teamId.
130         */
131        int getIndexForTeamId(int teamId) throws
               IDNotRecognisedException{
132            int index =-1;
133            if (allTeams.size() == 0){
134                throw new IDNotRecognisedException("No Team exists
                       with that ID");
135            }
136            for (int i=0; i<allTeams.size();i++){
```

```
137             if (allTeams.get(i).getId()==teamId){
138                 index = i;
139                 break;
140             }
141         }
142         if (index == -1){
143             throw new IDNotRecognisedException("No rider exists
                    with that ID");
144         }
145         return index;
146     }
147     /**
148      * @param teamId int - A teamId of a team to search for its
              object. If the ID doesn't exist
             IDNotRecognisedException is thrown.
149      * @throws IDNotRecognisedException teamId int - If the ID
             doesn't exist IDNotRecognisedException is thrown.
150      * @return A Team object with the teamId parsed.
151      */
152     Team getTeam(int teamId) throws IDNotRecognisedException{
153         int teamIndex = getIndexForTeamId(teamId);
154         return allTeams.get(teamIndex);
155     }
156     /**
157      * @param riderId int - A riderId of a team to search for
             its object. If the ID doesn't exist
             IDNotRecognisedException is thrown.
158      * @throws IDNotRecognisedException riderId int - If the ID
              doesn't exist IDNotRecognisedException is thrown.
159      * @return A Rider object with the riderId parsed.
160      */
161     Rider getRider(int riderId) throws IDNotRecognisedException
            {
162         int riderIndex = getIndexForRiderId(riderId);
163         return allRiders.get(riderIndex);
164     }
165     void setAllTeams(ArrayList<Team> allTeams){
166
167         RiderManager.allTeams = allTeams;
168         if (allTeams.size() != 0){
169         Team lastTeam = allTeams.get(allTeams.size()-1);
170         Team.teamTopId = lastTeam.getId()+1;
171         }
172     }
173     void setAllRiders(ArrayList<Rider> allRiders){
174         RiderManager.allRiders = allRiders;
175         if (allRiders.size() != 0){
176             Rider lastRider = allRiders.get(allRiders.size()-1)
                    ;
177             Rider.ridersTopId = lastRider.getRiderId()+1;
```

```
178             }
179         }
180         int [] getRiderIds(){
181             int[] riderIdArray = new int[allRiders.size()];
182             int count = 0;
183             for (Rider rider : RiderManager.allRiders){
184                 riderIdArray[count] = rider.getRiderId();
185                 count++;
186 
187             }
188             return riderIdArray;
189         }
190 
191     }
```