1 CyclingPortal.java

```java
package cycling;

import java.util.Arrays;
import java.util.Comparator;
import java.util.HashMap;
import java.io.IOException;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.ArrayList;
import java.io.ObjectOutputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.FileInputStream;

/**
 * CyclingPortal implements CyclingPortalInterface; contains methods for
 * handling the following classes: Race, Stage, Segment, RiderManager (and in
 * turn Rider and Team), and Result.
 * These classes are used manage races and their subdivisions, teams and their
 * riders, and to calculate and assign points from riders' results.
 * Also contains methods for saving and loading (Mini) CyclingPortalInterface to
 * and from a file.
 *
 * @author Ethan Ray & Thomas Newbold
 * @version 1.1
 *
 */
public class CyclingPortal implements CyclingPortalInterface {
    public RiderManager riderManager = new RiderManager();

    @Override
    public int[] getRaceIds() {
        return Race.getAllRaceIds();
    }

    @Override
    public int createRace(String name, String description)
        throws IllegalNameException, InvalidNameException {
        Race r = new Race(name, description);
        return r.getRaceId();
    }

```

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

```
            IDNotRecognisedException {
82          Race.removeStage(stageId);
83      }
84
85      @Override
86      public int addCategorizedClimbToStage(int stageId, Double
            location, SegmentType type, Double averageGradient,
87              Double length) throws IDNotRecognisedException,
                    InvalidLocationException,
                    InvalidStageStateException,
88              InvalidStageTypeException {
89          return Stage.addSegmentToStage(stageId, location, type,
                 averageGradient, length);
90      }
91
92      @Override
93      public int addIntermediateSprintToStage(int stageId, double
             location) throws IDNotRecognisedException,
94              InvalidLocationException,
                    InvalidStageStateException,
                    InvalidStageTypeException {
95          // adds stage with type SPRINT, and length 0.0
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     @Override
125     public int[] getTeams() {
126         return riderManager.getTeams();
127     }
128
129     @Override
130     public int[] getTeamRiders(int teamId) throws
            IDNotRecognisedException {
131         return riderManager.getTeamRiders(teamId);
132     }
133
134     @Override
135     public int createRider(int teamID, String name, int
            yearOfBirth) throws IDNotRecognisedException,
            IllegalArgumentException {
136         return riderManager.createRider(teamID, name,
                yearOfBirth);
137     }
138
139     @Override
140     public void removeRider(int riderId) throws
            IDNotRecognisedException {
141         riderManager.removeRider(riderId);
142     }
143
144     @Override
145     public void registerRiderResultsInStage(int stageId, int
            riderId, LocalTime... checkpoints)
146             throws IDNotRecognisedException,
                    DuplicatedResultException,
                    InvalidCheckpointsException,
147                 InvalidStageStateException {
148         if(Stage.getStageState(stageId).equals(StageState.
                BUILDING)) {
149             throw new InvalidStageStateException("stage is not
                    waiting for results");
150         } else if(Stage.getSegments(stageId).length+2 !=
                checkpoints.length) {
151             throw new InvalidCheckpointsException("checkpoint
                    count mismatch");
152         }
153         try {
154             Result.getResult(stageId, riderId);
155             throw new DuplicatedResultException("result already
```

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

```
196     }
197
198     @Override
199     public void deleteRiderResultsInStage(int stageId, int
            riderId) throws IDNotRecognisedException {
200         Stage.getStage(stageId);
201         riderManager.getRider(riderId);
202         // above should throw exceptions if IDs are not in
                system
203         Result.removeResult(stageId, riderId);
204     }
205
206     @Override
207     public int[] getRidersRankInStage(int stageId) throws
            IDNotRecognisedException {
208         Result[] results = Result.getResultsInStage(stageId);
209         int[] riderRanks = new int[results.length];
210         Arrays.fill(riderRanks, -1); // 0 may be a rider id
211         for(Result r : results) {
212             for(int i=0; i<riderRanks.length; i++) {
213                 if(riderRanks[i] == -1) {
214                     riderRanks[i] = r.getRiderId();
215                     break;
216                 } else if(r.getTotalElasped().isBefore(Result.
                        getResult(stageId, riderRanks[i]).
                        getTotalElasped())) {
217                     // add id at position i, move other ids
                            down
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                 // adjusted splits measured from adjusted start
                        time
245                 LocalTime adjustedSplit;
246                 for(int j=0; j<adjustedTimes.length; j++) {
247                     adjustedSplit = Result.getElapsed(adjustedTimes
                            [j], checkpoints[j]);
248                     // adjusted per segment
249                     out[i] = out[i].plusHours(adjustedSplit.getHour
                            ());
250                     out[i] = out[i].plusMinutes(adjustedSplit.
                            getMinute());
251                     out[i] = out[i].plusSeconds(adjustedSplit.
                            getSecond());
252                     out[i] = out[i].plusNanos(adjustedSplit.getNano
                            ());
253                 }
254             }
255             return out;
256         }
257 
258         @Override
259         public int[] getRidersPointsInStage(int stageId) throws
                IDNotRecognisedException {
260             StageType type = Stage.getStageType(stageId);
261             int[] points = new int[Result.getResultsInStage(stageId
                    ).length];
262             int[] distribution = new int[15];
263             // distributions from https://en.wikipedia.org/wiki/
                    Points_classification_in_the_Tour_de_France
264             // The points to be awarded in order for the stage
265             switch(type) {
266                 case FLAT:
267                     distribution = new int
                            []{50,30,20,18,16,14,12,10,8,7,6,5,4,3,2};
268                     break;
269                 case MEDIUM_MOUNTAIN:
270                     distribution = new int
                            []{30,25,22,19,17,15,13,11,9,7,6,5,4,3,2};
271                     break;
272                 case HIGH_MOUNTAIN:
273                     distribution = new int
                            []{20,17,15,13,11,10,9,8,7,6,5,4,3,2,1};
274                     break;
```

```
275                 case TT:
276                     distribution = new int
                            []{20,17,15,13,11,10,9,8,7,6,5,4,3,2,1};
277                     break;
278             }
279             for(int i=0; i<Math.min(points.length, distribution.
                    length); i++) {
280                 points[i] = distribution[i];
281             }
282             // check for SPRINT checkpoints
283             distribution = new int
                    []{20,17,15,13,11,10,9,8,7,6,5,4,3,2,1};
284             ArrayList<Integer> ridersArray = new ArrayList<Integer
                    >();
285             for(int r : getRidersRankInStage(stageId)) {
                    ridersArray.add(r); }
286             // converts RRIS from int[] to ArrayList
287             int[] segments = Stage.getSegments(stageId);
288             Result[] results;
289             for(int s=0; s<segments.length; s++) {
290                 if(Segment.getSegmentType(segments[s]).equals(
                        SegmentType.SPRINT)) {
291                     // get ranks for segment
292                     results = Result.getResultsInStage(stageId);
293                     int[] riderRanks = new int[results.length];
294                     Arrays.fill(riderRanks, -1); // 0 may be a
                            rider id
295                     for(Result r : results) {
296                         for(int i=0; i<riderRanks.length; i++) {
297                             if(riderRanks[i] == -1) {
298                                 riderRanks[i] = r.getRiderId();
299                                 break;
300                             } else if(r.getCheckpoints()[s].
                                    isBefore(Result.getResult(stageId,
                                    riderRanks[i]).getCheckpoints()[s]))
                                     {
301                                 int temp;
302                                 int prev = r.getRiderId();
303                                 for(int j=i; j<riderRanks.length; j
                                        ++) {
304                                     temp = riderRanks[j];
305                                     riderRanks[j] = prev;
306                                     prev = temp;
307                                     if(prev == -1) {
308                                         break;
309                                     }
310                                 }
311                                 break;
312                             }
313                         }
```

```
314                 }
315                 // adds points to position of rider in overall
                        ranking
316                 for(int i=0; i<Math.min(points.length,
                        distribution.length); i++) {
317                     int overallPos = ridersArray.indexOf(
                            riderRanks[i]);
318                     if(overallPos<points.length && overallPos
                            !=-1) {
319                         points[overallPos] += distribution[i];
320                     }
321                 }
322             }
323         }
324         return points;
325     }
326
327     @Override
328     public int[] getRidersMountainPointsInStage(int stageId)
            throws IDNotRecognisedException {
329         Result[] results = Result.getResultsInStage(stageId);
330         // All results refering to the stage with id *stageId*
331         int[] riders = getRidersRankInStage(stageId);
332         // An int array of rider ids, from first to last
333         int[] segments = Stage.getSegments(stageId);
334         // An int array of the segment ids in the stage
335         int[] points = new int[riders.length];
336         // The int in position i is the number of points to be
                awarded to the rider with id riders[i]
337         for(int s=0; s<segments.length; s++) {
338             SegmentType type = Segment.getSegmentType(segments[
                    s]);
339             int[] distribution = new int[1];
340             // The points to be awarded in order for the
                    segment
341             switch(type) {
342                 case C4:
343                     distribution = new int[]{1};
344                     break;
345                 case C3:
346                     distribution = new int[]{2,1};
347                     break;
348                 case C2:
349                     distribution = new int[]{5,3,2,1};
350                     break;
351                 case C1:
352                     distribution = new int[]{10,8,6,4,2,1};
353                     break;
354                 case HC:
355                     distribution = new int
```

```
                        []{20,15,12,10,8,6,4,2};
356                     break;
357                 case SPRINT:
358             }
359             // get ranks for segment
360             int[] riderRanks = new int[results.length];
361             Arrays.fill(riderRanks, -1); // 0 may be a rider id
362             for(Result r : results) {
363                 for(int i=0; i<riderRanks.length; i++) {
364                     if(riderRanks[i] == -1) {
365                         riderRanks[i] = r.getRiderId();
366                         break;
367                     } else if(r.getCheckpoints()[s].isBefore(
                            Result.getResult(stageId, riderRanks[i])
                            .getCheckpoints()[s])) {
368                         int temp;
369                         int prev = r.getRiderId();
370                         for(int j=i; j<riderRanks.length; j++)
                                {
371                             temp = riderRanks[j];
372                             riderRanks[j] = prev;
373                             prev = temp;
374                             if(prev == -1) {
375                                 break;
376                             }
377                         }
378                         break;
379                     }
380                 }
381             }
382             // adds points to position of rider in overall
                    ranking
383             ArrayList<Integer> ridersArray = new ArrayList<
                    Integer>();
384             for(int r : riders) { ridersArray.add(r); }
385             for(int i=0; i<Math.min(points.length, distribution
                    .length); i++) {
386                 int overallPos = ridersArray.indexOf(riderRanks
                        [i]);
387                 if(overallPos<points.length && overallPos!=-1)
                        {
388                     points[overallPos] += distribution[i];
389                 }
390             }
391         }
392         return points;
393     }
394
395     @Override
396     public void eraseCyclingPortal() {
```

```
397         Team.teamNames.clear();
398         Team.teamTopId = 0;
399         Rider.ridersTopId = 0;
400 
401         RiderManager.allRiders.clear();
402         RiderManager.allTeams.clear();
403 
404 
405         Race.allRaces.clear();
406         Race.removedIds.clear();
407         Race.loadId();
408 
409         Segment.allSegments.clear();
410         Segment.removedIds.clear();
411         Segment.loadId();
412 
413         Stage.allStages.clear();
414         Stage.removedIds.clear();
415         Stage.loadId();
416 
417         Result.allResults.clear();
418     }
419 
420     @Override
421     public void saveCyclingPortal(String filename) throws
            IOException {
422         try {
423             FileOutputStream fos = new FileOutputStream(
                    filename);
424             ObjectOutputStream oos = new ObjectOutputStream(fos
                    );
425             ArrayList<ArrayList> allObj = new ArrayList<>();
426             allObj.add(RiderManager.allTeams);
427             allObj.add(RiderManager.allRiders);
428             allObj.add(Stage.allStages);
429             allObj.add(Stage.removedIds);
430             allObj.add(Race.allRaces);
431             allObj.add(Race.removedIds);
432             allObj.add(Result.allResults);
433             allObj.add(Segment.allSegments);
434             allObj.add(Segment.removedIds);
435 
436             oos.writeObject(allObj);
437 
438             oos.flush();
439             oos.close();
440         } catch (IOException ex) {
441             ex.printStackTrace();
442         }
443 
```

```
444     }
445
446     @Override
447     /**
448      * @param raceId filename String - A vaild race Id to get
           the the riders rank in order.
449      * @throws IOException name String - Has to be non-null or
           IllegalArgumentException is thrown.
450      * @throws ClassNotFoundException if in the save file their
           is a non specified Class.
451      */
452     public void loadCyclingPortal(String filename) throws
         IOException, ClassNotFoundException {
453         try {
454
455             FileInputStream fis = new FileInputStream(filename)
                ;
456             ObjectInputStream ois = new ObjectInputStream(fis);
457             ArrayList<Object> allObjects = new ArrayList<>();
458             ArrayList<Team> allTeams = new ArrayList<>();
459             ArrayList<Rider> allRiders = new ArrayList<>();
460             ArrayList<Result> allResults = new ArrayList<Result
                >();
461             ArrayList<Race> allRaces = new ArrayList<Race>();
462             ArrayList<Stage> allStages = new ArrayList<Stage>()
                ;
463             ArrayList<Segment> allSegments = new ArrayList<
                Segment>();
464             ArrayList<Integer> removedIds = new ArrayList<>();
465
466             Class<?> classFlag = null;
467
468             allObjects = (ArrayList) ois.readObject(); //Get
                all objects in the filename
469             for (Object tempObj : allObjects){
470                 ArrayList Objects = (ArrayList) tempObj; //
                    Convert all the objects to an arrayList
471             for (Object obj : Objects){
472                 if (classFlag != null){
473                     if (obj.getClass() != classFlag && obj.
                        getClass() != Integer.class){ //Get our
                        defiend classs and add all the removeds
                        ids back from save
474                         if (classFlag == Race.class){
475                             Race.removedIds = removedIds;
476                         }
477                         if (classFlag == Segment.class){
478                             Segment.removedIds = removedIds;
479                         }
480                         if (classFlag == Stage.class){
```

```
481                         Stage.removedIds = removedIds;
482                     }
483                     classFlag = null;
484                     removedIds.clear();
485
486
487                 }
488                 else{
489                     Integer removedId = (Integer) obj; //
                        Fix removedIds
490                     removedIds.add(removedId);
491
492                 }
493             }
494             String objClass = obj.getClass().getName();
495             System.out.println(objClass);
                    // Do all the magic of added each class
                we need
496             if (obj.getClass() == Rider.class){
497                 Rider newRider = (Rider) obj;
498                 allRiders.add(newRider);
499                 System.out.println("NEW RIDER");
500             }
501             if (obj.getClass() == Team.class){
502                 Team newTeam = (Team) obj;
503                 allTeams.add(newTeam);
504                 System.out.println("NEW TEAM");
505             }
506             if (obj.getClass() == Result.class){
507                 Result newResult = (Result) obj;
508                 allResults.add(newResult);
509                 System.out.println("NEW RESULT");
510             }
511             if (obj.getClass() == Stage.class){
512                 Stage newStage = (Stage) obj;
513                 allStages.add(newStage);
514                 System.out.println("NEW STAGE");
515                 classFlag = Stage.class;
516             }
517             if (obj.getClass() == Race.class){
518                 Race newRace = (Race) obj;
519                 allRaces.add(newRace);
520                 System.out.println("NEW Race");
521                 classFlag = Race.class;
522             }
523             if (obj.getClass() == Segment.class){
524                 Segment newSeg = (Segment) obj;
525                 allSegments.add(newSeg);
526                 System.out.println("NEW SEGMENT");
527                 classFlag = Segment.class;
```

```
528                 }
529
530
531                 System.out.println(obj.getClass());
532             }
533         }
534         if (classFlag == Race.class){ // Add last removed IDs
535             Race.removedIds = removedIds;
536         }
537         if (classFlag == Segment.class){
538             Segment.removedIds = removedIds;
539         }
540         if (classFlag == Stage.class){
541             Stage.removedIds = removedIds;
542         }
543
544             this.riderManager.setAllTeams(allTeams); //Load all
                    team reiders ids etc etc.
545             this.riderManager.setAllRiders(allRiders);
546             Race.allRaces = allRaces;
547             Race.loadId();
548             Stage.allStages = allStages;
549             Stage.loadId();
550             Segment.allSegments = allSegments;
551             Segment.loadId();
552             Result.allResults = allResults;
553             ois.close();
554
555         }
556         catch (Exception ex) {
557             ex.printStackTrace();
558         }
559
560     }
561
562     @Override
563     public void removeRaceByName(String name) throws
            NameNotRecognisedException {
564         boolean found = false;
565         for (int raceId : Race.getAllRaceIds()){
566             try {
567                 if (name == Race.getRaceName(raceId)){
568                     Race.removeRace(raceId);
569                 }
570             }
571             catch(Exception c){
572                 assert(false); // Exception will not throw by
                        for each condition
573                 // This try catch is easier than moving
                        exceptions to CyclingPortal level
```

```
574                 }
575
576             }
577             if (!found){ throw new NameNotRecognisedException("Name
                    not in System.");}
578         }
579
580         @Override
581         public LocalTime[] getGeneralClassificationTimesInRace(int
                raceId) throws IDNotRecognisedException {
582             Race currentRace = Race.getRace(raceId); //get race
583             int[] stageIds = currentRace.getStages(); // get all
                    stages in the race
584             int[] riderIds = this.riderManager.getRiderIds(); //
                    get all rider Ids in the system
585             HashMap<Integer,Long> riderElaspedTime = new HashMap<
                    Integer,Long>(); //Rider Id : totalTime (long)
586             for (int riderId : riderIds){
587                 riderElaspedTime.put(riderId,0L); // (Map all
                        RiderId's to a long of total time starting of 0)
                         (Total time is in nano seconds hence the long)
588             }
589             for (int stageId : stageIds){
590                 Result[] temp = Result.getResultsInStage(stageId);
                        //Get all the resuts in the each stage
591                 for(Result result: temp){ //get the rider id of
                        each stage and their timeTaken.
592                     int riderId = result.getRiderId();
593                     LocalTime getTotalElasped = result.
                            getTotalElasped();
594                     long timeTaken = getTotalElasped.toNanoOfDay();
595                     Long newTime = (Long)riderElaspedTime.get(
                            riderId)+timeTaken;
596                     riderElaspedTime.put(riderId,newTime); //
                            update the hash map
597                 }
598
599             }
600             long[][] riderTimePos = new long[riderIds.length][2];
                    //2d arr to hold riderId and timetaken
601             int count = 0;
602             for (int riderId : riderIds){
603                 Long finalRiderTime = riderElaspedTime.get(riderId)
                        ;// ## -> [[time,riderId],....] sort by time!
604                 riderTimePos[count][0] = riderId;
605                 riderTimePos[count][1] = finalRiderTime;
606                 count++;
607             }
608             Arrays.sort(riderTimePos, Comparator.comparingDouble(o
                    -> o[1]));
```

```
609             LocalTime[] finalTimes = new LocalTime[riderIds.length
                    ];
610             count = 0;
611             for (long[] items : riderTimePos){
612                 finalTimes[count]= LocalTime.ofNanoOfDay(items[1]);
613                 count++;
614             }
615
616
617
618             return finalTimes;
619         }
620
621         @Override
622         public int[] getRidersPointsInRace(int raceId) throws
                IDNotRecognisedException {
623             ArrayList<Integer> order = new ArrayList<Integer>();
624             for(int riderId : getRidersGeneralClassificationRank(
                    raceId)) {
625                 order.add(riderId); // converts GCR from int[] to
                        ArrayList
626             }
627             int[] out = new int[order.size()];
628             int[] stageRank, stagePoints;
629             for(int stageId : Race.getStages(raceId)) {
630                 stageRank = getRidersRankInStage(stageId);
631                 stagePoints = getRidersPointsInStage(stageId);
632                 for(int i=0; i<stageRank.length; i++) {
633                     out[order.indexOf(stageRank[i])] += stagePoints
                            [i];
634                     // orders points from stagePoints by order
                            using stageRank
635                 }
636             }
637             return out;
638         }
639
640         @Override
641         public int[] getRidersMountainPointsInRace(int raceId)
                throws IDNotRecognisedException {
642             ArrayList<Integer> order = new ArrayList<Integer>();
643             for(int riderId : getRidersGeneralClassificationRank(
                    raceId)) {
644                 order.add(riderId); // converts GCR from int[] to
                        ArrayList
645             }
646             int[] out = new int[order.size()];
647             int[] stageRank, stagePoints;
648             for(int stageId : Race.getStages(raceId)) {
649                 stageRank = getRidersRankInStage(stageId);
```

```
650             stagePoints = getRidersMountainPointsInStage(
                    stageId);
651             for(int i=0; i<stageRank.length; i++) {
652                 out[order.indexOf(stageRank[i])] += stagePoints
                        [i];
653                 // orders points from stagePoints by order
                        using stageRank
654             }
655         }
656         return out;
657     }
658     @Override
659     public int[] getRidersGeneralClassificationRank(int raceId)
            throws IDNotRecognisedException {
660         Race currentRace = Race.getRace(raceId); //get race
661         int[] stageIds = currentRace.getStages(); // get all
                stages in the race
662         int[] riderIds = this.riderManager.getRiderIds(); //
                get all rider Ids in the system
663         HashMap<Integer,Long> riderElaspedTime = new HashMap<
                Integer,Long>(); //Rider Id : totalTime (long)
664         for (int riderId : riderIds){
665             riderElaspedTime.put(riderId,0L); // (Map all
                    RiderId's to a long of total time starting of 0)
                     (Total time is in nano seconds hence the long)
666         }
667         for (int stageId : stageIds){
668             Result[] temp = Result.getResultsInStage(stageId);
                    //Get all the resuts in the each stage
669             for(Result result: temp){ //get the rider id of
                    each stage and their timeTaken.
670                 int riderId = result.getRiderId();
671                 LocalTime getTotalElasped = result.
                        getTotalElasped();
672                 long timeTaken = getTotalElasped.toNanoOfDay();
673                 Long newTime = (Long)riderElaspedTime.get(
                        riderId)+timeTaken;
674                 riderElaspedTime.put(riderId,newTime); //
                        update the hash map
675             }
676
677         }
678         long[][] riderTimePos = new long[riderIds.length][2];
                //2d arr to hold riderId and timetaken
679         int count = 0;
680         for (int riderId : riderIds){
681             Long finalRiderTime = riderElaspedTime.get(riderId)
                    ;// ## -> [[time,riderId],....] sort by time!
682             riderTimePos[count][0] = riderId;
683             riderTimePos[count][1] = finalRiderTime;
```

```
684             count++;
685         }
686         Arrays.sort(riderTimePos, Comparator.comparingDouble(o
                -> o[1])); // Sort by time.
687         int[] finalPos = new int[riderIds.length];
688         count = 0;
689         for (long[] items : riderTimePos){
690             finalPos[count]= (int)items[0];
691             count++;
692         }
693 
694         return finalPos;
695     }
696 
697     @Override
698     public int[] getRidersPointClassificationRank(int raceId)
            throws IDNotRecognisedException {
699         ArrayList<Integer> order = new ArrayList<Integer>();
700         for(int riderId : getRidersGeneralClassificationRank(
                raceId)) {
701             order.add(riderId); // converts GCR from int[] to
                    ArrayList
702         }
703         int[] points = getRidersPointsInRace(raceId);
704         int[] out = new int[order.size()];
705         for(int i=0; i<out.length; i++) {
706             int maxPoints = -1;
707             int nextId = -1;
708             for(int j=0; j<order.size(); j++) {
709                 int id = order.get(j);
710                 if(id<0) { continue; }
711                 if(points[id] > maxPoints) {
712                     maxPoints = points[j];
713                     nextId = id;
714                 }
715                 // fetches highest points
716             }
717             if(maxPoints < 0) {
718                 break;
719             } else {
720                 out[i] = nextId;
721                 order.set(order.indexOf(nextId), -1);
722                 // adds id to out, removes from check (order)
723             }
724         }
725         return out;
726     }
727 
728     @Override
729     public int[] getRidersMountainPointClassificationRank(int
```

```
        raceId) throws IDNotRecognisedException {
730         ArrayList<Integer> order = new ArrayList<Integer>();
731         for(int riderId : getRidersGeneralClassificationRank(
                raceId)) {
732             order.add(riderId); // converts GCR from int[] to
                    ArrayList
733         }
734         int[] points = getRidersMountainPointsInRace(raceId);
735         int[] out = new int[order.size()];
736         for(int i=0; i<out.length; i++) {
737             int maxPoints = -1;
738             int nextId = -1;
739             for(int j=0; j<order.size(); j++) {
740                 int id = order.get(j);
741                 if(id<0) { continue; }
742                 if(points[id] > maxPoints) {
743                     maxPoints = points[j];
744                     nextId = id;
745                 }
746                 // fetches highest points
747             }
748             if(maxPoints < 0) {
749                 break;
750             } else {
751                 out[i] = nextId;
752                 order.set(order.indexOf(nextId), -1);
753                 // adds id to out, removes from check (order)
754             }
755         }
756         return out;
757     }
758 }
```

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                 assert(false); // exception will be thrown in
                        this case when stage is created
175         }
```

```
176             String name = this.stageName;
177             String description = this.stageDescription;
178             String length = Double.toString(this.stageLength);
179             DateTimeFormatter formatter = DateTimeFormatter.
                    ofPattern("HH:hh dd-MM-yyyy");
180             String startTime = this.stageStartTime.format(formatter
                    );
181             String list = this.segmentIds.toString();
182             String type;
183             switch (this.stageType) {
184                 case FLAT:
185                     type = "Flat";
186                     break;
187                 case MEDIUM_MOUNTAIN:
188                     type = "Medium Mountain";
189                     break;
190                 case HIGH_MOUNTAIN:
191                     type = "High Mountain";
192                     break;
193                 case TT:
194                     type = "Time Trial";
195                     break;
196                 default:
197                     type = "null type";
198                     assert(false); // exception will be thrown in
                            this case when stage is created
199             }
200             return String.format("Stage[%s](%s): %s (%s); %s; %skm;
                     %s; SegmentIds=%s;",
201                                  id, state, name, type, description
                                      , length,
202                                  startTime, list);
203         }
204
205         /**
206          * @param id The ID of the stage
207          * @return A string representation of the stage instance
208          * @throws IDNotRecognisedException If no stage exists with
                    the requested ID
209          */
210         public static String toString(int id) throws
                IDNotRecognisedException {
211             return getStage(id).toString();
212         }
213
214         /**
215          * @return The integer stageId for the stage instance
216          */
217         public int getStageId() { return this.stageId; }
218
```

```
219     /**
220      * @return The state of the stage instance
221      */
222     public StageState getStageState() { return this.stageState;
            }
223
224     /**
225      * @param id The ID of the stage
226      * @return The state of the stage instance
227      * @throws IDNotRecognisedException If no stage exists with
               the requested ID
228      */
229     public static StageState getStageState(int id) throws
230                                             IDNotRecognisedException
                                                    {
231         return getStage(id).getStageState();
232     }
233     /**
234      * @return The string raceName for the stage instance
235      */
236     public String getStageName() { return this.stageName; }
237
238     /**
239      * @param id The ID of the stage
240      * @return The string stageName for the stage with the
               associated id
241      * @throws IDNotRecognisedException If no stage exists with
               the requested ID
242      */
243     public static String getStageName(int id) throws
            IDNotRecognisedException {
244         return getStage(id).stageName;
245     }
246
247     /**
248      * @return The string stageDescription for the stage
               instance
249      */
250     public String getStageDescription() { return this.
            stageDescription; }
251
252     /**
253      * @param id The ID of the stage
254      * @return The string stageDescription for the stage with
               the associated id
255      * @throws IDNotRecognisedException If no stage exists with
               the requested ID
256      */
257     public static String getStageDescription(int id) throws
258                                              IDNotRecognisedException
```

```
                                                        {
259             return getStage(id).stageDescription;
260         }
261
262         /**
263          * @return The length of the stage instance
264          */
265         public double getStageLength() { return this.stageLength; }
266
267         /**
268          * @param id The ID of the stage
269          * @return The length of the stage instance
270          * @throws IDNotRecognisedException If no stage exists with
                 the requested ID
271          */
272         public static double getStageLength(int id) throws
                IDNotRecognisedException {
273             return getStage(id).stageLength;
274         }
275
276         /**
277          * @return The start time for the stage instance
278          */
279         public LocalDateTime getStageStartTime() { return this.
                stageStartTime; }
280
281         /**
282          * @param id The ID of the stage
283          * @return The start time for the stage instance
284          * @throws IDNotRecognisedException If no stage exists with
                 the requested ID
285          */
286         public static LocalDateTime getStageStartTime(int id)
                throws
287                                                     IDNotRecognisedException
                                                         {
288             return getStage(id).stageStartTime;
289         }
290
291         /**
292          * @return The type of the stage instance
293          */
294         public StageType getStageType() { return this.stageType; }
295
296         /**
297          * @param id The ID of the stage
298          * @return The type of the stage instance
299          * @throws IDNotRecognisedException If no stage exists with
                 the requested ID
300          */
```

```
301     public static StageType getStageType(int id) throws
            IDNotRecognisedException {
302         return getStage(id).getStageType();
303     }
304
305     /**
306      * @return An integer array of segment IDs for the stage
             instance
307      */
308     public int[] getSegments() {
309         int length = this.segmentIds.size();
310         int[] segmentIdsArray = new int[length];
311         for(int i=0; i<length; i++) {
312             segmentIdsArray[i] = this.segmentIds.get(i);
313         }
314         return segmentIdsArray;
315     }
316
317     /**
318      * @param id The ID of the stage
319      * @return An integer array of segment IDs for the stage
             instance
320      * @throws IDNotRecognisedException If no stage exists with
              the requested ID
321      */
322     public static int[] getSegments(int id) throws
            IDNotRecognisedException {
323         Stage stage = getStage(id);
324         int length = stage.segmentIds.size();
325         int[] segmentIdsArray = new int[length];
326         for(int i=0; i<length; i++) {
327             segmentIdsArray[i] = stage.segmentIds.get(i);
328         }
329         return segmentIdsArray;
330     }
331
332     /**
333      * Updates the stage state from building to waiting for
             results.
334      *
335      * @throws InvalidStageStateException If the stage is
             already waiting for results
336      */
337     public void updateStageState() throws
            InvalidStageStateException {
338         if(this.stageState.equals(StageState.WAITING)) {
339             throw new InvalidStageStateException("stage is
                    already waiting for results");
340         } else if(this.stageState.equals(StageState.BUILDING))
                {
```

```
341             this.stageState = StageState.WAITING;
342         }
343     }
344
345     /**
346      * Updates the stage state from building to waiting for
            results.
347      *
348      * @param id The ID of the stage to be updated
349      * @throws IDNotRecognisedException If no stage exists with
            the requested ID
350      * @throws InvalidStageStateException If the stage is
            already waiting for results
351      */
352     public static void updateStageState(int id) throws
        IDNotRecognisedException,
353                                         InvalidStageStateException
                                                {
354         getStage(id).updateStageState();
355     }
356
357     /**
358      * @param name The new name for the stage instance
359      */
360     public void setStageName(String name) {
361         this.stageName = name;
362     }
363
364     /**
365      * @param id The ID of the stage to be updated
366      * @param name The new name for the stage instance
367      * @throws IDNotRecognisedException If no stage exists with
            the requested ID
368      */
369     public static void setStageName(int id, String name) throws
370                                     IDNotRecognisedException {
371         getStage(id).setStageName(name);
372     }
373
374     /**
375      * @param description The new description for the stage
            instance
376      */
377     public void setStageDescription(String description) {
378         this.stageDescription = description;
379     }
380
381     /**
382      * @param id The ID of the stage to be updated
383      * @param description The new description for the stage
```

```
             instance
384          * @throws IDNotRecognisedException If no stage exists with
                 the requested ID
385          */
386         public static void setStageDescription(int id, String
                description) throws
387                                                 IDNotRecognisedException
                                                        {
388             getStage(id).setStageDescription(description);
389         }
390
391         /**
392          * @param length The new length for the stage instance
393          */
394         public void setStageLength(double length) {
395             this.stageLength = length;
396         }
397
398         /**
399          * @param id The ID of the stage to be updated
400          * @param length The new length for the stage instance
401          * @throws IDNotRecognisedException If no stage exists with
                 the requested ID
402          */
403         public static void setStageLength(int id, double length)
                throws
404                                           IDNotRecognisedException
                                                  {
405             getStage(id).stageLength = length;
406         }
407
408         /**
409          * @param startTime The new start time for the stage
                 instance
410          */
411         public void setStageStartTime(LocalDateTime startTime) {
412             this.stageStartTime = startTime;
413         }
414
415         /**
416          * @param id The ID of the stage to be updated
417          * @param startTime The new start time for the stage
                 instance
418          * @throws IDNotRecognisedException If no stage exists with
                 the requested ID
419          */
420         public static void setStageStartTime(int id, LocalDateTime
                startTime)
421                                               throws
                                                    IDNotRecognisedException
```

```
                                                    {
422             getStage(id).stageStartTime = startTime;
423         }
424
425         /**
426          * Creates a new stage and adds the ID to the stageIds
                array.
427          *
428          * @param location The location of the new segment
429          * @param type The type of the new segment
430          * @param averageGradient The average gradient of the new
                segment
431          * @param length The length (in km) of the new segment
432          * @throws InvalidLocationException If the segment finishes
                outside of the
433          *                                      bounds of the stage
434          * @throws InvalidStageStateException If the segment state
                is waiting for
435          *                                       results
436          * @throws InvalidStageTypeException If the stage type is a
                time-trial
437          *                                      (cannot contain
                segments)
438          */
439         public int addSegmentToStage(double location, SegmentType
                type,
440                                      double averageGradient, double
                                             length) throws
441                                      InvalidLocationException,
442                                      InvalidStageStateException,
443                                      InvalidStageTypeException {
444             if(location > this.getStageLength()) {
445                 throw new InvalidLocationException("segment
                        finishes outside of stage bounds");
446             }
447             if(this.getStageState().equals(StageState.WAITING)) {
448                 throw new InvalidStageStateException("stage is
                        waiting for results");
449             }
450             if(this.getStageType().equals(StageType.TT)) {
451                 throw new InvalidStageTypeException("time trial
                        stages cannot contain segments");
452             }
453             Segment newSegment = new Segment(location, type,
                    averageGradient, length);
454             this.segmentIds.add(newSegment.getSegmentId());
455             return newSegment.getSegmentId();
456         }
457
458         /**
```

```
459         * Creates a new stage and adds the ID to the stageIds
                array.
460         *
461         * @param id The ID of the stage to which the segment will
                be added
462         * @param location The location of the new segment
463         * @param type The type of the new segment
464         * @param averageGradient The average gradient of the new
                segment
465         * @param length The length (in km) of the new segment
466         * @throws IDNotRecognisedException If no stage exists with
                the requested ID
467         * @throws InvalidLocationException If the segment finishes
                outside of the
468         *                                      bounds of the stage
469         * @throws InvalidStageStateException If the segment state
                is waiting for
470         *                                       results
471         * @throws InvalidStageTypeException If the stage type is a
                time-trial
472         *                                       (cannot contain
                segments)
473         */
474        public static int addSegmentToStage(int id, double location
               , SegmentType type,
475                                            double averageGradient,
                                                    double length)
                                                   throws
476                                            IDNotRecognisedException
                                                   ,
477                                            InvalidLocationException
                                                   ,
478                                            InvalidStageStateException
                                                   ,
479                                            InvalidStageTypeException
                                                    {
480            return getStage(id).addSegmentToStage(location, type,
                   averageGradient, length);
481        }
482
483        /**
484         * Removes a segmentId from the array of segmentIds for a
                stage instance,
485         * as well as from the static array of all segments in the
                Segment class.
486         *
487         * @param segmentId The ID of the segment to be removed
488         * @throws IDNotRecognisedException If no segment exists
                with the requested
489         *                                      ID
```

```
490         */
491        private void removeSegmentFromStage(int segmentId) throws
492                                                IDNotRecognisedException
                                                        {
493            if(this.segmentIds.contains(segmentId)) {
494                this.segmentIds.remove(segmentId);
495                Segment.removeSegment(segmentId);
496            } else {
497                throw new IDNotRecognisedException("segmentID not
                       found in race");
498            }
499        }
500
501        /**
502         * Removes a segmentId from the array of segmentIds for a
                stage instance,
503         * as well as from the static array of all segments in the
                Segment class.
504         *
505         * @param id The ID of the stage to which the segment will
                be removed
506         * @param segmentId The ID of the segment to be removed
507         * @throws IDNotRecognisedException If no segment exists
                with the requested
508         *                                      ID
509         */
510        public static void removeSegmentFromStage(int id, int
               segmentId) throws
511                                                   IDNotRecognisedException
                                                           {
512            getStage(id).removeSegmentFromStage(segmentId);
513        }
514
515        /**
516         * Removes a segmentId from the array of segmentIds for a
                stage instance,
517         * as well as from the static array of all segments in the
                Segment class.
518         *
519         * @param segmentId The ID of the segment to be removed
520         * @throws IDNotRecognisedException If no segment exists
                with the requested
521         *                                      ID
522         */
523        public static void removeSegment(int segmentId) throws
               IDNotRecognisedException {
524            for(Stage stage : allStages) {
525                if(stage.segmentIds.contains(segmentId)) {
526                    stage.removeSegmentFromStage(segmentId);
527                    break;
```

```
528                 }
529             }
530         }
531 }
```

4 StageState.java

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

5 Segment.java

```
1  package cycling;
2
3  import java.io.Serializable;
4  import java.util.ArrayList;
5
6  /**
7   * Segment encapsulates race segments
8   *
9   * @author Thomas Newbold
10  * @version 2.0
11  *
12  */
13 public class Segment implements Serializable {
14     // Static class attributes
15     private static int idMax = 0;
```

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

```
57          */
58         public static int[] getAllSegmentIds() {
59             int length = Segment.allSegments.size();
60             int[] segmentIdsArray = new int[length];
61             int i = 0;
62             for(Segment segment : allSegments) {
63                 segmentIdsArray[i] = segment.getSegmentId();
64                 i++;
65             }
66             return segmentIdsArray;
67         }
68 
69         /**
70          * @param segmentId The ID of the segment instance to
                remove
71          * @throws IDNotRecognisedException If no segment exists
                with the requested
72          *                                  ID
73          */
74         public static void removeSegment(int segmentId) throws
75                                          IDNotRecognisedException {
76             boolean removed = Segment.removedIds.contains(segmentId
                );
77             if(segmentId<Segment.idMax && segmentId >= 0 && !
                removed) {
78                 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                 assert(false); // exception will be thrown in
                        this case when segment is created
148         }
149         String averageGrad = Double.toString(this.
                segmentAverageGradient);
150         String length = Double.toString(this.segmentLength);
151         return String.format("Segment[%s]: %s; %skm; Location=%
                s; Gradient=%s;",
152                              id, type, length, location,
                                    averageGrad);
153     }
154
155     /**
156      * @param id The ID of the segment
157      * @return A string representation of the segment instance
158      * @throws IDNotRecognisedException If no segment exists
                with the requested
159      *                                  ID
160      */
161     public static String toString(int id) throws
            IDNotRecognisedException {
162         return getSegment(id).toString();
163     }
164
165     /**
166      * @return The integer segmentId for the segment instance
167      */
168     public int getSegmentId() { return this.segmentId; }
169
170     /**
171      * @return The integer representing the location of the
                segment instance
172      */
173     public double getSegmentLocation() { return this.
            segmentLocation; }
174
175     /**
176      * @param id The ID of the segment
177      * @return The integer representing the location of the
                segment instance
178      * @throws IDNotRecognisedException If no segment exists
                with the requested
179      *                                  ID
180      */
181     public static double getSegmentLocation(int id) throws
182                                             IDNotRecognisedException
                                                    {
183         return getSegment(id).segmentLocation;
184     }
```

```
185
186         /**
187          * @return The type of the segment instance
188          */
189         public SegmentType getSegmentType() { return this.
                segmentType; }
190
191         /**
192          * @param id The ID of the segment
193          * @return The type of the segment instance
194          * @throws IDNotRecognisedException If no segment exists
                    with the requested
195          *                                     ID
196          */
197         public static SegmentType getSegmentType(int id) throws
198                                                     IDNotRecognisedException
                                                         {
199             return getSegment(id).segmentType;
200         }
201
202         /**
203          * @return The average gradient of the segment instance
204          */
205         public double getSegmentAverageGradient() {
206             return this.segmentAverageGradient;
207         }
208
209         /**
210          * @param id The ID of the segment
211          * @return The average gradient of the segment instance
212          * @throws IDNotRecognisedException If no segment exists
                    with the requested
213          *                                     ID
214          */
215         public static double getSegmentAverageGradient(int id)
                throws
216                                                           IDNotRecognisedException
                                                               {
217             return getSegment(id).segmentAverageGradient;
218         }
219
220         /**
221          * @return The length of the segment instance
222          */
223         public double getSegmentLength() { return this.
                segmentLength; }
224
225         /**
226          * @param id The ID of the segment
227          * @return The length of the segment instance
```

```
228         * @throws IDNotRecognisedException If no segment exists
                with the requested
229         *                                    ID
230         */
231        public static double getSegmentLength(int id) throws
               IDNotRecognisedException {
232            return getSegment(id).segmentLength;
233        }
234
235        /**
236         * @param location The new location for the segment
                instance
237         */
238        public void setSegmentLocation(double location) {
239            this.segmentLocation = location;
240        }
241
242        /**
243         * @param id The ID of the segment to be updated
244         * @param location The new location for the segment
                instance
245         * @throws IDNotRecognisedException If no segment exists
                with the requested
246         *                                    ID
247         */
248        public static void setSegmentLocation(int id, double
               location) throws
249                                               IDNotRecognisedException
                                                      {
250            getSegment(id).setSegmentLocation(location);
251        }
252
253        /**
254         * @param type The new type for the segment instance
255         */
256        public void setSegmentType(SegmentType type) {
257            this.segmentType = type;
258        }
259
260        /**
261         * @param id The ID of the segment to be updated
262         * @param type The new type for the segment instance
263         * @throws IDNotRecognisedException If no segment exists
                with the requested
264         *                                    ID
265         */
266        public static void setSegmentType(int id, SegmentType type)
               throws
267                                           IDNotRecognisedException
                                                  {
```

```
268             getSegment(id).setSegmentType(type);
269         }
270 
271         /**
272          * @param averageGradient The new average gradient for the
                segment instance
273          */
274         public void setSegmentAverageGradient(double
                averageGradient) {
275             this.segmentAverageGradient = averageGradient;
276         }
277 
278         /**
279          * @param id The ID of the segment to be updated
280          * @param averageGradient The new average gradient for the
                segment instance
281          * @throws IDNotRecognisedException If no segment exists
                with the requested
282          *                                     ID
283          */
284         public static void setSegmentAverageGradient(int id, double
                averageGradient)
285                                                         throws
                                                             IDNotRecognisedException
                                                              {
286             getSegment(id).setSegmentAverageGradient(
                    averageGradient);
287         }
288 
289         /**
290          * @param length The new length for the segment instance
291          */
292         public void setSegmentLength(double length) {
293             this.segmentLength = length;
294         }
295 
296         /**
297          * @param id The ID of the segment to be updated
298          * @param length The new length for the segment instance
299          * @throws IDNotRecognisedException If no segment exists
                with the requested
300          *                                     ID
301          */
302         public static void setSegmentLength(int id, double length)
                throws
303                                                 IDNotRecognisedException
                                                      {
304             getSegment(id).setSegmentLength(length);
305         }
306 }
```

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            double nanos = a.until(b, ChronoUnit.NANOS);
164            nanos = nanos%Math.pow(10, 9);
165            return LocalTime.of(hours%24, minuites%60, seconds%60,
                    (int)nanos);
166        }
167
168        /**
```

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

7 Team.java

```
1   package cycling;
2   import java.io.Serializable;
3   import java.util.ArrayList;
4   /**
5    * Team Class holds the teamId,name,description and riderIds
         belonging to that team.
6    *
7    *
8    * @author Ethan Ray
```

```
 * @version 1.0
 *
 */

public class Team implements Serializable {
    public static ArrayList<String> teamNames = new ArrayList
        <>();
    public static int teamTopId = 0;

    private int teamID;
    private String name;
    private String description;
    private ArrayList<Integer> riderIds = new ArrayList<>();


    /**
     * @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.
     * @param description String - A description for the team.
     * @throws IllegalNameException name String - Is a
         duplicate name of any other Team, IllegalNameException
         will be thrown.
     * @throws InvailNameException name String - If the name is
          null, empty, has more than 30 characters, or has white
          spaces will throw InvaildNameException.
     */
    public Team(String name, String description) throws
        IllegalNameException, InvalidNameException
    {
        if (name == "" || name.length()>30 || name.contains(" "
            )){
            throw new InvalidNameException("Team name cannot be
                 empty, longer than 30 characters , or has white
                 spaces.");
        }
        for (int i = 0;i<teamNames.size();i++){
            if (teamNames.get(i) == name){
                throw new IllegalNameException("That team name
                    already exsists!");
            }
        }

        teamNames.add(name);
        this.teamID = teamTopId++;
        this.name = name;
        this.description = description;
    }
    /**
     * @param rider Rider - A rider to add to the team.
```

```
47          */
48         public void addRider(Rider rider){
49
50             this.riderIds.add(rider.getRiderId());
51         }
52         /**
53          * @param riderId int - A riderId to be removed from the
                team.
54          */
55         public void removeRiderId(int riderId){
56             for (int i =0;i<this.riderIds.size();i++){
57                 if (this.riderIds.get(i)==riderId){
58                     this.riderIds.remove(i);
59                     break;
60                 }
61             }
62         }
63         /**
64          * @return An Array of integers - which are the riderIds in
                that team.
65          */
66         public int[] getRiderIds(){
67             int [] currentRiderIds = new int[this.riderIds.size()];
68             for (int i=0; i<this.riderIds.size();i++){
69                 currentRiderIds[i]=this.riderIds.get(i);
70             }
71             return currentRiderIds;
72         }
73         /**
74          * @return A Integer - teamId of the team.
75          */
76         public int getId(){
77             return this.teamID;
78         }
79         /**
80          * @return A String - Name of the team.
81          */
82         public String getTeamName(){
83             return this.name;
84         }
85         /**
86          * @return A String - The description of the team.
87          */
88         public String getDescription(){
89             return this.description;
90         }
91     }
```

8 Rider.java

```
1  package cycling;
2
3  import java.io.Serializable;
4
5  /**
6   * Rider Class holds the riders teamId,riderId,name and
        yearOfBirth
7   *
8   *
9   * @author Ethan Ray
10  * @version 1.0
11  *
12  */
13
14
15 public class Rider implements Serializable {
16     public static int ridersTopId;
17     private int riderId;
18     private int teamID;
19     private String name;
20     private int yearOfBirth;
21
22
23     /**
24      * @param teamID int - A team Id that the rider will belong
            too
25      * @param name String - A name for the rider, Has to be non
            -null or IllegalArgumentException is thrown.
26      * @param yearOfBirth int - A year that the rider was born
            in. Has to be above 1900 or IllegalArgumentException is
            thrown.
27      * @throws IllegalArgumentException name String - Has to be
            non-null or IllegalArgumentException is thrown.
28      * @throws IllegalArgumentException yearOfBirth int - A
            year that the rider was born in. Has to be above 1900
            or IllegalArgumentException is thrown.
29      */
30     public Rider(int teamID, String name, int yearOfBirth)
           throws IllegalArgumentException
31     {
32         this.riderId = ridersTopId++;
33         this.teamID = teamID;
34         if (name == "" || name == null){
35             throw new IllegalArgumentException("Illegal name
                   entered for rider");
36         }
37         this.name = name;
```

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

9 RiderManager.java

```
1   package cycling;
2
3   import java.io.Serializable;
4   import java.util.ArrayList;
5
6   public class RiderManager implements Serializable{
7       public static ArrayList<Rider> allRiders = new ArrayList
            <>();
8       public static ArrayList<Team> allTeams = new ArrayList<>();
9
10
11      /**
12       * @param teamID int - A team Id that the rider will belong
```

```
             too. If the ID doesn't exist IDNotRecognisedException
            is thrown.
13          * @param name String - A name for the rider, Has to be non
             -null or IllegalArgumentException is thrown.
14          * @param yearOfBirth int - A year that the rider was born
             in. Has to be above 1900 or IllegalArgumentException is
             thrown.
15          * @return riderId of the rider created.
16          * @throws IDNotRecognisedException teamId int - If the ID
             doesn't exist IDNotRecognisedException is thrown.
17          * @throws IllegalArgumentException yearOfBirth int - A
             year that the rider was born in. Has to be above 1900
             or IllegalArgumentException is thrown.
18          */
19         int createRider(int teamID, String name, int yearOfBirth)
            throws IDNotRecognisedException,IllegalArgumentException
            {
20              int teamIndex = getIndexForTeamId(teamID);
21              Rider newRider = new Rider(teamID,name,yearOfBirth);
22              allRiders.add(newRider);
23              Team ridersTeam = allTeams.get(teamIndex);
24              ridersTeam.addRider(newRider);
25              return newRider.getRiderId();
26         }
27         /**
28          * @param riderId int - A riderId of a rider to be removed.
             If the ID doesn't exist IDNotRecognisedException is
            thrown.
29          * @throws IDNotRecognisedException riderId int - If the ID
             doesn't exist IDNotRecognisedException is thrown.
30          */
31         void removeRider(int riderId) throws
            IDNotRecognisedException
32         {
33              int riderIndex = getIndexForRiderId(riderId);
34              int teamId = allRiders.get(riderIndex).getRiderTeamId()
                 ;
35              int teamIndex = getIndexForTeamId(teamId);
36              Team riderTeam = allTeams.get(teamIndex);
37              riderTeam.removeRiderId(riderId);
38              allRiders.remove(riderIndex);
39         }
40         /**
41          * @param riderId int - A riderId of a rider to be searced
             for. If the ID doesn't exist IDNotRecognisedException
            is thrown.
42          * @throws IDNotRecognisedException riderId int - If the ID
             doesn't exist IDNotRecognisedException is thrown.
43          * @return An int which is the index that maps to the
            riderId.
```

```
44          */
45      int getIndexForRiderId(int riderId) throws
            IDNotRecognisedException{
46          int index =-1;
47          if (allRiders.size() == 0){
48              throw new IDNotRecognisedException("No rider exists
                    with that ID");
49          }
50          for (int i=0; i<allRiders.size();i++){
51              if (allRiders.get(i).getRiderId()==riderId){
52                  index = i;
53                  break;
54              }
55          }
56          if (index == -1){
57              throw new IDNotRecognisedException("No rider exists
                    with that ID");
58          }
59          return index;
60      }
61      /**
62       * @param name String - A name for the team, , If the name
             is null, empty, has more than 30 characters, or has
             white spaces will throw InvaildNameException.
63       * @param description String - A description for the team.
64       * @throws IllegalNameException name String - Is a
             duplicate name of any other Team, IllegalNameException
             will be thrown.
65       * @throws InvailNameException name String - If the name is
              null, empty, has more than 30 characters, or has white
              spaces will throw InvaildNameException.
66       */
67      int createTeam(String name, String description) throws
            IllegalNameException, InvalidNameException{
68          Team newTeam = new Team(name,description);
69          allTeams.add(newTeam);
70          return newTeam.getId();
71      }
72      /**
73       * @param teamId int - A teamId of a rider to be removed.
             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     /**
166      * @param allTeams ArrayList<Team> - A list of all teams to
             be set.
167      */
168     void setAllTeams(ArrayList<Team> allTeams){
169
170         RiderManager.allTeams = allTeams;
171         if (allTeams.size() != 0){
172         Team lastTeam = allTeams.get(allTeams.size()-1);
173         Team.teamTopId = lastTeam.getId()+1;
174         }
175     }
176     /**
177      * @param allRider ArrayList<Rider> - A list of all riders
            to be set.
178      */
179     void setAllRiders(ArrayList<Rider> allRiders){
180         RiderManager.allRiders = allRiders;
181         if (allRiders.size() != 0){
182             Rider lastRider = allRiders.get(allRiders.size()-1)
                    ;
183             Rider.ridersTopId = lastRider.getRiderId()+1;
184         }
185     }
186     /**
187      * @return The list of all rider Ids
188      */
189     int [] getRiderIds(){
190         int[] riderIdArray = new int[allRiders.size()];
191         int count = 0;
192         for (Rider rider : RiderManager.allRiders){
193             riderIdArray[count] = rider.getRiderId();
194             count++;
195
196         }
197         return riderIdArray;
198     }
199
200 }
```

10 RaceTestApp.java

```
1  package cycling;
2
3  import java.time.LocalDateTime;
```

```
4
5  public class RaceTestApp {
6      public static void main(String[] args) throws
           IllegalNameException,
7                                InvalidNameException,
                                     InvalidLengthException,
8                                IDNotRecognisedException,
                                     InvalidStageStateException,
9                                InvalidStageTypeException,
                                     InvalidLocationException {
10         System.out.println("instanstiating...\n");
11         Race r1 = new Race("France", "Tour de France");
12         r1.addStageToRace("Monaco", "Road Race", 10.0,
               LocalDateTime.of(2022, 1, 9, 13, 0), StageType.TT);
13         r1.addStageToRace("Nice", "Mountain Road", 25.0,
               LocalDateTime.of(2022, 1, 11, 9, 0), StageType.
               HIGH_MOUNTAIN);
14         int[] stageIDs = r1.getStages();
15         Stage s1 = Stage.getStage(stageIDs[1]);
16         s1.addSegmentToStage(5.0, SegmentType.SPRINT, 1.0, 5.0)
               ;
17         s1.addSegmentToStage(15.0, SegmentType.C3, 4.5, 10.0);
18         s1.addSegmentToStage(21.0, SegmentType.C1, 8.0, 6.0);
19         s1.addSegmentToStage(25.0, SegmentType.C2, 10.0, 4.0);
20         int[] segmentIds = s1.getSegments();
21         // instance states
22         System.out.println(r1.toString());
23         System.out.println(Stage.getStage(stageIDs[0]).toString
               ());
24         System.out.println(s1.toString());
25         for(int id : segmentIds) {
26             System.out.println(Segment.getSegment(segmentIds[id
                   ]));
27         }
28         System.out.println("\nremoving instances...  [stage 0;
               segment 3]\n");
29         Race.removeStage(0);
30         s1 = Stage.getStage(r1.getStages()[0]);
31         Stage.removeSegment(3);
32         segmentIds = s1.getSegments();
33         // instance states
34         System.out.println(r1.toString());
35         System.out.println(s1.toString());
36         for(int id : segmentIds) {
37             System.out.println(Segment.getSegment(segmentIds[id
                   ]));
38         }
39     }
40 }
```

11 ResultTestApp.java

```
1  package cycling;
2
3  import java.time.LocalDateTime;
4  import java.time.LocalTime;
5  import java.util.Arrays;
6
7  public class ResultTestApp {
8      public static void main(String[] args) throws
           IllegalNameException, InvalidNameException,
9                              IDNotRecognisedException,
                                   InvalidLengthException,
                                   InvalidLocationException,
10                             InvalidStageStateException,
                                   InvalidStageTypeException,
                                   DuplicatedResultException,
11                             InvalidCheckpointsException {
12         CyclingPortal portal = new CyclingPortal();
13         // creating races, adding stages/segments
14         int race = portal.createRace("France", "Tour de France"
               );
15         int[] stages = new int[2];
16         stages[0] = portal.addStageToRace(race, "Monaco", "Road
               Race", 10.0,
17                                           LocalDateTime.of
                                                 (2022, 1, 9, 13,
                                                 0),
18                                           StageType.TT);
19         stages[1] = portal.addStageToRace(race, "Nice", "
               Mountain Road", 25.0,
20                                           LocalDateTime.of
                                                 (2022, 1, 11, 9,
                                                 0),
21                                           StageType.
                                                 HIGH_MOUNTAIN);
22         int[] segments = new int[4];
23         segments[0] = portal.addIntermediateSprintToStage(
               stages[1], 5.0);
24         segments[1] = portal.addCategorizedClimbToStage(stages
               [1], 15.0, SegmentType.C3, 4.5, 10.0);
25         segments[2] = portal.addCategorizedClimbToStage(stages
               [1], 21.0, SegmentType.C1, 8.0, 6.0);
26         segments[3] = portal.addCategorizedClimbToStage(stages
               [1], 25.0, SegmentType.C2, 10.0, 4.0);
27         portal.concludeStagePreparation(stages[0]);
28         portal.concludeStagePreparation(stages[1]);
29         // creating teams and riders
30         int[] teams = new int[2];
```

```
31          teams[0] = portal.createTeam("EthansTeam", "Ethans Team
                for Racing!");
32          teams[1] = portal.createTeam("ThomasTeam", "Thomas Team
                for Racing?");
33          int[] riders = new int[4];
34          riders[0] = portal.createRider(teams[0], "Ethan", 2003)
                ;
35          riders[1] = portal.createRider(teams[1], "Thomas",
                2003);
36          riders[2] = portal.createRider(teams[0], "Jeff", 2002);
37          riders[3] = portal.createRider(teams[1], "Bob", 2002);
38          // registering times
39          portal.registerRiderResultsInStage(stages[0], riders
                [0], new LocalTime[]{LocalTime.of(13,0,0),LocalTime.
                of(13,18,51)});
40          portal.registerRiderResultsInStage(stages[0], riders
                [1], new LocalTime[]{LocalTime.of(13,0,0),LocalTime.
                of(13,19,8)});
41          portal.registerRiderResultsInStage(stages[0], riders
                [2], new LocalTime[]{LocalTime.of(13,0,0),LocalTime.
                of(13,19,42)});
42          portal.registerRiderResultsInStage(stages[0], riders
                [3], new LocalTime[]{LocalTime.of(13,0,0),LocalTime.
                of(13,19,11)});
43
44          portal.registerRiderResultsInStage(stages[1], riders
                [0], new LocalTime[]{LocalTime.of(9,0,0),LocalTime.
                of(9,10,0), LocalTime.of(9,32,44),LocalTime.of
                (9,43,2), LocalTime.of(9,58,30),LocalTime.of
                (9,58,30)});
45          portal.registerRiderResultsInStage(stages[1], riders
                [1], new LocalTime[]{LocalTime.of(9,0,0),LocalTime.
                of(9,9,59), LocalTime.of(9,32,33),LocalTime.of
                (9,43,21), LocalTime.of(9,59,20),LocalTime.of
                (9,59,20)});
46          portal.registerRiderResultsInStage(stages[1], riders
                [2], new LocalTime[]{LocalTime.of(9,0,0),LocalTime.
                of(9,10,10), LocalTime.of(9,33,1),LocalTime.of
                (9,45,0), LocalTime.of(10,1,4),LocalTime.of(10,1,4)
                });
47          portal.registerRiderResultsInStage(stages[1], riders
                [3], new LocalTime[]{LocalTime.of(9,0,0),LocalTime.
                of(9,10,11), LocalTime.of(9,32,58),LocalTime.of
                (9,44,40), LocalTime.of(10,0,11),LocalTime.of
                (10,0,11)});
48          // fetching points
49          //System.out.println(Result.getResult(0, 0).toString())
                ;
50          Result[] stage1 = Result.getResultsInStage(stages[0]);
51          Result[] stage2 = Result.getResultsInStage(stages[1]);
```

```
52          //System.out.println(stage1.length);
53          for(Result r : stage1) {
54              System.out.println(r.toString());
55          }
56          System.out.println("");
57          for(Result r : stage2) {
58              System.out.println(r.toString());
59          }
60          System.out.println("\nStage 0:");
61          System.out.println(Arrays.toString(portal.
                getRidersRankInStage(stages[0])));
62          System.out.println(Arrays.toString(portal.
                getRidersPointsInStage(stages[0])));
63          System.out.println(Arrays.toString(portal.
                getRidersMountainPointsInStage(stages[0])));
64          System.out.println("Adjusted elapsed times in stage 0:"
                );
65          System.out.println(Arrays.toString(portal.
                getRankedAdjustedElapsedTimesInStage(stages[0])));
66          System.out.println("Stage 1:");
67          System.out.println(Arrays.toString(portal.
                getRidersRankInStage(stages[1])));
68          System.out.println(Arrays.toString(portal.
                getRidersPointsInStage(stages[1])));
69          System.out.println(Arrays.toString(portal.
                getRidersMountainPointsInStage(stages[1])));
70
71          System.out.println("\nRace Classification:");
72          System.out.println(Arrays.toString(portal.
                getRidersPointClassificationRank(race)));
73          System.out.println(Arrays.toString(portal.
                getRidersPointsInRace(race)));
74          System.out.println("Race Classification (Mountain):");
75          System.out.println(Arrays.toString(portal.
                getRidersMountainPointClassificationRank(race)));
76          System.out.println(Arrays.toString(portal.
                getRidersMountainPointsInRace(race)));
77      }
78  }
```

12 RiderManagerTestApp.java

```
1   package cycling;
2   import java.util.Arrays;
3
4   public class RiderManagerTestApp {
5       public static void main(String[] args) throws
            IllegalNameException,InvalidNameException,
            IDNotRecognisedException{
```

```
6           RiderManager myRiderManager = new RiderManager();
7           int testTeamId=myRiderManager.createTeam("EthansTeam",
                "Ethans Team for Racing!!!");
8           System.out.println("Creating team EthansTeam");
9           int testTeamId2=myRiderManager.createTeam("EthansTeam2"
                , "Ethans Team2 for Racing!!!");
10          System.out.println("Creating team EthansTeam2");
11          myRiderManager.createRider(testTeamId, "Ethan", 2003);
12          System.out.println("Adding Rider Ethan to EthansTeam (
                Born 2003)");
13          myRiderManager.createRider(testTeamId, "Thomas", 2002);
14          System.out.println("Adding Rider Thomas to EthansTeam2
                (Born 2002)");
15          myRiderManager.createRider(testTeamId2, "Jeff", 2002);
16          System.out.println("Adding Rider Jeff to EthansTeam2 (
                Born 2002)");
17          printInfo(myRiderManager,testTeamId);
18          //int testTeamId2=myRiderManager.createTeam("EthansTeam
                ", "Ethans Team for Racing!!!"); // Throws
                IllegalName Exception as expected
19          System.out.println("Removing Rider with ID of 0 (Ethan)
                ");
20          myRiderManager.removeRider(0);
21          printInfo(myRiderManager,testTeamId);
22          myRiderManager.createRider(testTeamId, "Ethan", 2003);
23          System.out.println("Adding Rider Ethan to EthansTeam (
                Born 2003)");
24          //myRiderManager.createRider(testTeamId, "", 2003); //
                Throws Illegal Arg Exception as expected
25          //myRiderManager.createRider(testTeamId, "OldEthan",
                1800); // Throws Illegal Arg Expection as edxpected
26          //myRiderManager.createRider(testTeamId, null, 2003);
                //Throws Illegal Arg Exception as expected
27          printInfo(myRiderManager,testTeamId);
28          System.out.println("Deleting Team 0 & Riders in that
                team");
29          myRiderManager.removeTeam(testTeamId);
30          printInfo(myRiderManager, testTeamId2);
31
32
33
34      }
35      public static void printInfo(RiderManager myRiderManager,
            int testTeamId) throws IDNotRecognisedException{
36          System.out.println("
                ---------------------------------------------");
37          System.out.println("RiderNames:"+Arrays.toString(
                myRiderManager.getRidersNames()));
38          System.out.println("Id of Riders in teamId "+testTeamId
                +":"+Arrays.toString(myRiderManager.getTeamRiders(
```

```
            testTeamId)));
39          System.out.println("TeamNames:"+Arrays.toString(
                myRiderManager.getTeamsNames()));
40          System.out.println("Id of Teams:"+Arrays.toString(
                myRiderManager.getTeams()));
41          System.out.println("
                ------------------------------------------------");
42      }
43  }
```

13 CyclingPortalTestApp.java

```
1   package cycling;
2
3
4   import java.io.IOException;
5   import java.time.LocalDateTime;
6
7   public class CyclingPortalTestApp {
8       public static void main(String[] args) throws
            IDNotRecognisedException,IllegalNameException,
            InvalidNameException,IOException,ClassNotFoundException{
9           if (true){
10          CyclingPortal testCyclingPortal = new CyclingPortal();
11          int teamId = testCyclingPortal.createTeam("TESTTEAM", "
                SOME DESC");
12          testCyclingPortal.createRace("TESTRACE", "SOME RACE
                DESC");
13          int removeID = testCyclingPortal.createRace("REMOVEME",
                 "SOME RACE DESC");
14          testCyclingPortal.removeRaceById(removeID);
15          testCyclingPortal.createRider(teamId, "somename", 2001)
                ;
16          testCyclingPortal.saveCyclingPortal("testsave.data");
17          testCyclingPortal.eraseCyclingPortal();
18
19          }
20          CyclingPortal testCyclingPortal2 = new CyclingPortal();
21          testCyclingPortal2.loadCyclingPortal("testsave.data");
22          for (int teamID : testCyclingPortal2.riderManager.
                getTeams()){
23              System.out.println(teamID);
24          }
25          for (int raceID : testCyclingPortal2.getRaceIds()){
26              System.out.println(raceID);
27              System.out.println(testCyclingPortal2.
                    viewRaceDetails(raceID));
28          }
29          System.out.println("RIDER 0 ID : "+testCyclingPortal2.
```

```
            riderManager.getRider(0));
30          int testriderId = testCyclingPortal2.riderManager.
                createRider(0, "testrider", 2000);
31          System.out.println("new rider ID SHOULD BE 1 not 0 :"+
                testriderId);
32          System.out.println("RIDER IDS");
33          for (int riderId : testCyclingPortal2.riderManager.
                getTeamRiders(0)){
34              System.out.println(riderId);
35          }
36
37          System.out.println("RACE REMOVEDID");
38          for (int removedID : Race.removedIds){
39              System.out.println(removedID);
40          }
41
42          testCyclingPortal2.eraseCyclingPortal();
43
44          System.out.println("--------------------------------
                ERASED");
45
46          for (int teamID : testCyclingPortal2.riderManager.
                getTeams()){
47              System.out.println(teamID);
48          }
49          for (int raceID : testCyclingPortal2.getRaceIds()){
50              System.out.println(raceID);
51              System.out.println(testCyclingPortal2.
                    viewRaceDetails(raceID));
52          }
53
54          System.out.println("RACE REMOVEDID");
55          for (int removedID : Race.removedIds){
56              System.out.println(removedID);
57          }
58          //testCyclingPortal2.riderManager.getRider(100); errors
                 correctly
59          int racetest1 = testCyclingPortal2.createRace("
                RACETEST1", "The coolest race ever!");
60          try {
61              testCyclingPortal2.addStageToRace(racetest1, "
                    Stage1", "DESC TEST", 10.0, LocalDateTime.now(),
                     StageType.FLAT);
62          } catch (InvalidLengthException e) {
63              e.printStackTrace();
64          }
65          //System.out.println(testCyclingPortal2.
                getRidersGeneralClassificationRank(0));
66
67
```

```
68
69              }
70
71      }
```