```java
package cycling;

import java.io.*;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class CyclingPortal implements CyclingPortalInterface {

    private List<Race> raceList = new ArrayList<>();
    private List<Team> teamList = new ArrayList<>();

    /**
     * Get the races currently created in the platform.
     *
     * @return An array of race IDs in the system or an empty array if none exists.
     */
    public int[] getRaceIds() {
        int[] raceIds = new int[raceList.size()];
        for (int i = 0; i < raceIds.length; i++) {
            raceIds[i] = raceList.get(i).getId();
        }
        return raceIds;
    }

    public int createRace(String name, String description) throws
IllegalNameException, InvalidNameException {
        validateName(name);

        Race race1 = getRace(name);
        if (race1 != null) {
            throw new IllegalNameException(name + " exists already.");
        }

        Race race = new Race(name, description);
        raceList.add(race);

        return race.getId();
    }

    public String viewRaceDetails(int raceId) throws IDNotRecognisedException {
        Race race = getRaceIfValidElseThrow(raceId);
        return race.toString();
    }

    public void removeRaceById(int raceId) throws IDNotRecognisedException {
        Race race = getRaceIfValidElseThrow(raceId);
        raceList.remove(race);
    }

    public int getNumberOfStages(int raceId) throws IDNotRecognisedException {
        Race race = getRaceIfValidElseThrow(raceId);
        return race.getStageList().size();
    }

```

```
58      public int addStageToRace(int raceId, String stageName, String description,
        double length, LocalDateTime startTime,
59                                StageType type)
60              throws IDNotRecognisedException, IllegalNameException,
        InvalidNameException, InvalidLengthException {
61          assert startTime != null;
62          if (length < 5) {
63              throw new InvalidLengthException("length can't be less than 5km.");
64          }
65
66          validateName(stageName);
67
68          Race race = getRaceIfValidElseThrow(raceId);
69
70          Stage stage1 = race.getStage(stageName);
71          if (stage1 != null) {
72              throw new IllegalNameException("name already exists in the platform.");
73          }
74
75          Stage stage = new Stage(stageName, description, length, startTime, type);
76          race.add(stage);
77
78          return stage.getId();
79      }
80
81      public int[] getRaceStages(int raceId) throws IDNotRecognisedException {
82          Race race = getRaceIfValidElseThrow(raceId);
83          int[] ids = race.getStageIds();
84          return ids;
85      }
86
87      public double getStageLength(int stageId) throws IDNotRecognisedException {
88          Stage stage = getStageFromAnyRace(stageId);
89          return stage.getLength();
90      }
91
92      public void removeStageById(int stageId) throws IDNotRecognisedException {
93          Stage stage = getStageFromAnyRace(stageId);
94          for (Race race : raceList) {
95              if (race.getStageList().contains(stage)) {
96                  race.remove(stage);
97                  break;
98              }
99          }
100     }
101
102     public int addCategorizedClimbToStage(int stageId, Double location, SegmentType
        type, Double averageGradient,
103                                           Double length) throws
        IDNotRecognisedException, InvalidLocationException, InvalidStageStateException,
104             InvalidStageTypeException {
105
106         Stage stage = getStageFromAnyRace(stageId);
107
108         if (stage.getLength() < location) {
109             throw new InvalidLocationException("location is out of bounds of the
        stage length.");
110         }
111
112         if (stage.getStageState().equals("waiting for results")) {
```

```
113             throw new InvalidStageStateException("stage is \"waiting for
    results\".");
114         }
115 
116         if (stage.getStageType() == StageType.TT) {
117             throw new InvalidStageTypeException("Time-trial stages cannot contain
    any segment.");
118         }
119 
120         Segment segment = new Segment(location, type, averageGradient, length);
121         stage.add(segment);
122 
123         return segment.getId();
124     }
125 
126     public int addIntermediateSprintToStage(int stageId, double location) throws
    IDNotRecognisedException,
127             InvalidLocationException, InvalidStageStateException,
    InvalidStageTypeException {
128         Stage stage = getStageFromAnyRace(stageId);
129 
130         if (stage.getLength() < location) {
131             throw new InvalidLocationException("location is out of bounds of the
    stage length.");
132         }
133 
134         if (stage.getStageState().equals("waiting for results")) {
135             throw new InvalidStageStateException("stage is \"waiting for
    results\".");
136         }
137 
138         if (stage.getStageType() == StageType.TT) {
139             throw new InvalidStageTypeException("Time-trial stages cannot contain
    any segment.");
140         }
141 
142         Segment segment = new Segment(location, SegmentType.SPRINT);
143         stage.add(segment);
144 
145         return segment.getId();
146     }
147 
148     public void removeSegment(int segmentId) throws IDNotRecognisedException,
    InvalidStageStateException {
149         Stage stage = getStageForSegmentIdFromAnyRace(segmentId);
150         if (stage.getStageState().equals("waiting for results")) {
151             throw new InvalidStageStateException("stage is \"waiting for
    results\".");
152         }
153 
154         Segment segmentToRemove = null;
155         List<Segment> segmentList = stage.getSegmentList();
156         for (Segment segment : segmentList) {
157             if (segment.getId() == segmentId) {
158                 segmentToRemove = segment;
159                 break;
160             }
161         }
162         if (segmentToRemove != null) {
163             stage.remove(segmentToRemove);
```

```
164         }
165     }
166 
167     public void concludeStagePreparation(int stageId) throws
IDNotRecognisedException, InvalidStageStateException {
168         Stage stage = getStageFromAnyRace(stageId);
169         if (stage.getStageState().equals("waiting for results")) {
170             throw new InvalidStageStateException("stage is \"waiting for
results\".");
171         }
172         stage.setStageState("waiting for results");
173     }
174 
175     public int[] getStageSegments(int stageId) throws IDNotRecognisedException {
176         Stage stage = getStageFromAnyRace(stageId);
177 
178         List<Segment> segmentList = stage.getSegmentList();
179         if (segmentList.isEmpty())
180             return new int[0];
181 
182         segmentList.sort((a, b) -> {
183             if (a.getLocation() < b.getLocation()) return -1;
184             if (a.getLocation() > b.getLocation()) return +1;
185             return 0;
186         });
187 
188         int[] ids = new int[segmentList.size()];
189         for (int i = 0; i < ids.length; i++) {
190             ids[i] = segmentList.get(i).getId();
191         }
192 
193         return ids;
194     }
195 
196     public int createTeam(String name, String description) throws
IllegalNameException, InvalidNameException {
197         for (Team team : teamList) {
198             if (team.getName().equals(name)) {
199                 throw new IllegalNameException("name already exists in the
platform.");
200             }
201         }
202 
203         validateName(name);
204 
205         Team team = new Team(name, description);
206         teamList.add(team);
207 
208         return team.getId();
209     }
210 
211     public void removeTeam(int teamId) throws IDNotRecognisedException {
212         Team team = getTeamIfValidElseThrow(teamId);
213         teamList.remove(team);
214     }
215 
216     public int[] getTeams() {
217         int[] ids = new int[teamList.size()];
218         for (int i = 0; i < ids.length; i++) {
219             ids[i] = teamList.get(i).getId();
```

```
220         }
221         return ids;
222     }
223 
224     public int[] getTeamRiders(int teamId) throws IDNotRecognisedException {
225         Team team = getTeamIfValidElseThrow(teamId);
226         List<Rider> riderList = team.getRiderList();
227         int[] ids = new int[riderList.size()];
228         for (int i = 0; i < ids.length; i++) {
229             ids[i] = riderList.get(i).getId();
230         }
231         return ids;
232     }
233 
234     public int createRider(int teamID, String name, int yearOfBirth) throws
235             IDNotRecognisedException, IllegalArgumentException {
236         Team team = getTeamIfValidElseThrow(teamID);
237 
238         if (name == null || yearOfBirth < 1900) {
239             throw new IllegalArgumentException("name of the rider is null or the
    year of birth is less than 1900.");
240         }
241 
242         Rider rider = new Rider(name, yearOfBirth);
243         team.add(rider);
244 
245         return rider.getId();
246     }
247 
248     public void removeRider(int riderId) throws IDNotRecognisedException {
249         Rider rider = getRiderIfValidElseThrow(riderId);
250         Team team = getTeamForRiderElseThrow(riderId);
251         team.remove(rider);
252 
253         /* When a rider is removed from the platform,
254          all of its results should be also removed.
255          Race results must be updated.*/
256 
257         for (Race race : raceList) {
258             for (Stage stage : race.getStageList()) {
259                 if (stage.hasResult(riderId)) {
260                     RiderResult result = stage.getResult(riderId);
261                     stage.removeResult(result);
262                 }
263             }
264         }
265     }
266 
267     public void registerRiderResultsInStage(int stageId, int riderId, LocalTime...
    checkpoints)
268             throws IDNotRecognisedException, DuplicatedResultException,
    InvalidCheckpointsException,
269             InvalidStageStateException {
270 
271         Stage stage = getStageFromAnyRace(stageId);
272         Rider rider = getRiderIfValidElseThrow(riderId);
273 
274         if (!stage.getStageState().equals("waiting for results")) {
275             throw new InvalidStageStateException("stage is not \"waiting for
    results\". Results can only be added to a stage while it is \"waiting for
```

```
results\".");
276         }
277
278         if (stage.getSegmentList().size() + 2 != checkpoints.length) {
279             throw new InvalidCheckpointsException("length of checkpoints is not
equal to n+2, where n is the number of segments in the stage; +2 represents the
start time and the finish time of the stage.");
280         }
281
282         if (stage.hasResult(riderId)) {
283             throw new DuplicatedResultException("rider has already a result for the
stage. Each rider can have only one result per stage.");
284         }
285
286         stage.addResult(stageId, riderId, checkpoints);
287     }
288
289     public LocalTime[] getRiderResultsInStage(int stageId, int riderId) throws
IDNotRecognisedException {
290         Stage stage = getStageFromAnyRace(stageId);
291         Rider rider = getRiderIfValidElseThrow(riderId);
292
293         if (!stage.hasResult(riderId)) {
294             return new LocalTime[0];
295         }
296
297         RiderResult result = stage.getResult(riderId);
298         return result.getSegmentTimes();
299     }
300
301     public LocalTime getRiderAdjustedElapsedTimeInStage(int stageId, int riderId)
throws
302             IDNotRecognisedException {
303
304         Stage stage = getStageFromAnyRace(stageId);
305         Rider rider = getRiderIfValidElseThrow(riderId);
306
307         if (!stage.hasResult(riderId)) {
308             return null;
309         }
310
311         RiderResult stageResult = stage.getResult(riderId);
312         if (stage.getStageType() == StageType.TT) {
313             return stageResult.getFinishTime();
314         }
315
316         LocalTime adjustedElapsedTime = stage.getAdjustedElapsedTime(riderId);
317
318         return adjustedElapsedTime;
319     }
320
321     public void deleteRiderResultsInStage(int stageId, int riderId) throws
IDNotRecognisedException {
322         Stage stage = getStageFromAnyRace(stageId);
323         Rider rider = getRiderIfValidElseThrow(riderId);
324         RiderResult result = stage.getResult(riderId);
325         stage.removeResult(result);
326     }
327
328     public int[] getRidersRankInStage(int stageId) throws IDNotRecognisedException {
```

```
329         Stage stage = getStageFromAnyRace(stageId);
330         if (stage.hasNoResult()) {
331             return new int[0];
332         }
333 
334         // A list of riders ID sorted by their elapsed time.
335         int[] ranks = stage.getRiderRanks();
336         return ranks;
337     }
338 
339     public LocalTime[] getRankedAdjustedElapsedTimesInStage(int stageId) throws
IDNotRecognisedException {
340         Stage stage = getStageFromAnyRace(stageId);
341         if (stage.hasNoResult()) {
342             return new LocalTime[0];
343         }
344 
345         LocalTime[] adjustedElapsedTimes = stage.getAdjustedElapsedTimes();
346         return adjustedElapsedTimes;
347     }
348 
349     public int[] getRidersPointsInStage(int stageId) throws IDNotRecognisedException
{
350 
351         Stage stage = getStageFromAnyRace(stageId);
352         if (stage.hasNoResult()) {
353             return new int[0];
354         }
355 
356         return stage.getRidersPointsInStage();
357     }
358 
359     public int[] getRidersMountainPointsInStage(int stageId) throws
IDNotRecognisedException {
360         Stage stage = getStageFromAnyRace(stageId);
361         if (stage.hasNoResult()) {
362             return new int[0];
363         }
364 
365         return stage.getRidersMountainPointsInStage();
366     }
367 
368     /**
369      * Method empties this MiniCyclingPortalInterface of its contents and resets all
370      * internal counters.
371      */
372     public void eraseCyclingPortal() {
373         raceList.clear();
374         teamList.clear();
375         Race.availableId = 1;
376         Rider.availableId = 1;
377         Segment.availableId = 1;
378         Stage.availableId = 1;
379         Team.availableId = 1;
380     }
381 
382     public void saveCyclingPortal(String filename) throws IOException {
383         //Saving of object in a file
384         FileOutputStream file = new FileOutputStream(filename);
385         ObjectOutputStream out = new ObjectOutputStream(file);
```

```
386
387         // Method for serialization of object
388         DataToSerializeDeserialize data = new DataToSerializeDeserialize();
389
390         data.availableRaceId = Race.availableId;
391         data.availableRiderId = Rider.availableId;
392         data.availableSegmentId = Segment.availableId;
393         data.availableStageId = Stage.availableId;
394         data.availableTeamId = Team.availableId;
395         data.raceList = raceList;
396         data.teamList = teamList;
397
398         out.writeObject(data);
399
400         out.close();
401         file.close();
402     }
403
404     public void loadCyclingPortal(String filename) throws IOException,
ClassNotFoundException {
405         // Reading the object from a file
406         FileInputStream file = new FileInputStream(filename);
407         ObjectInputStream in = new ObjectInputStream(file);
408
409         // Method for deserialization of object
410         DataToSerializeDeserialize data = (DataToSerializeDeserialize)
in.readObject();
411
412         Race.availableId = data.availableRaceId;
413         Rider.availableId = data.availableRiderId;
414         Segment.availableId = data.availableSegmentId;
415         Stage.availableId = data.availableStageId;
416         Team.availableId = data.availableTeamId;
417
418         raceList = data.raceList;
419         teamList = data.teamList;
420
421         in.close();
422         file.close();
423     }
424
425     public void removeRaceByName(String name) throws NameNotRecognisedException {
426         Race race = getRaceIfValidElseThrow(name);
427         raceList.remove(race);
428     }
429
430     public LocalTime[] getGeneralClassificationTimesInRace(int raceId) throws
IDNotRecognisedException {
431         Race race = getRaceIfValidElseThrow(raceId);
432         List<Stage> stageList = race.getStageList();
433         for (Stage stage : stageList) {
434             if (stage.hasNoResult()) {
435                 return new LocalTime[0];
436             }
437         }
438
439         Map<Integer, Integer> totalAdjustedTimes = new HashMap<>();
440
441         for (Stage stage : stageList) {
442
```

```
443             LocalTime[] adjustedElapsedTimes = stage.getAdjustedElapsedTimes();
444             int[] allIds = stage.getAllIds();
445 
446             for (int i = 0; i < allIds.length; i++) {
447                 int id = allIds[i];
448                 LocalTime time = adjustedElapsedTimes[i];
449                 totalAdjustedTimes.put(id, totalAdjustedTimes.getOrDefault(id, 0) +
time.toSecondOfDay());
450             }
451         }
452 
453         LocalTime[] ans = new LocalTime[totalAdjustedTimes.keySet().size()];
454 
455         int[] ridersGeneralClassificationRank =
getRidersGeneralClassificationRank(raceId);
456 
457         for (int i = 0, j = 0; j < ans.length && i <
ridersGeneralClassificationRank.length; i++, j++) {
458             int id = ridersGeneralClassificationRank[i];
459             int elapsedSeconds = totalAdjustedTimes.get(id);
460             int hour = elapsedSeconds / 3600;
461             int minute = (elapsedSeconds - hour * 3600) / 60;
462             int second = elapsedSeconds - hour * 3600 - minute * 60;
463             ans[j] = LocalTime.of(hour, minute, second);
464         }
465 
466         return ans;
467     }
468 
469     public int[] getRidersPointsInRace(int raceId) throws IDNotRecognisedException {
470         Race race = getRaceIfValidElseThrow(raceId);
471         List<Stage> stageList = race.getStageList();
472         for (Stage stage : stageList) {
473             if (stage.hasNoResult()) {
474                 return new int[0];
475             }
476         }
477 
478         Map<Integer, Integer> totalPoints = new HashMap<>();
479         for (Stage stage : race.getStageList()) {
480             int[] ridersPointsInStage = stage.getRidersPointsInStage();
481             int[] allIds = stage.getAllIds();
482             for (int i = 0; i < allIds.length; i++) {
483                 int id = allIds[i];
484                 int point = ridersPointsInStage[i];
485                 totalPoints.put(id, totalPoints.getOrDefault(id, 0) + point);
486             }
487         }
488 
489         // now sort by the total elapsed time.
490         int[] ans = new int[totalPoints.keySet().size()];
491         int[] ridersGeneralClassificationRank =
getRidersGeneralClassificationRank(raceId);
492 
493         for (int i = 0, j = 0; j < ans.length && i <
ridersGeneralClassificationRank.length; i++, j++) {
494             int id = ridersGeneralClassificationRank[i];
495             int totalPoint = totalPoints.get(id);
496             ans[j] = totalPoint;
497         }
```

```
498
499         return ans;
500     }
501
502     public int[] getRidersMountainPointsInRace(int raceId) throws
IDNotRecognisedException {
503         Race race = getRaceIfValidElseThrow(raceId);
504         List<Stage> stageList = race.getStageList();
505         for (Stage stage : stageList) {
506             if (stage.hasNoResult()) {
507                 return new int[0];
508             }
509         }
510
511         Map<Integer, Integer> totalPoints = new HashMap<>();
512         for (Stage stage : race.getStageList()) {
513             int[] ridersPointsInStage = stage.getRidersMountainPointsInStage();
514             int[] allIds = stage.getAllIds();
515             for (int i = 0; i < allIds.length; i++) {
516                 int id = allIds[i];
517                 int point = ridersPointsInStage[i];
518                 totalPoints.put(id, totalPoints.getOrDefault(id, 0) + point);
519             }
520         }
521
522         // now sort by the total elapsed time.
523         int[] ans = new int[totalPoints.keySet().size()];
524         int[] ridersGeneralClassificationRank =
getRidersGeneralClassificationRank(raceId);
525
526         for (int i = 0, j = 0; j < ans.length && i <
ridersGeneralClassificationRank.length; i++, j++) {
527             int id = ridersGeneralClassificationRank[i];
528             int totalPoint = totalPoints.get(id);
529             ans[j] = totalPoint;
530         }
531
532         return ans;
533     }
534
535     public int[] getRidersGeneralClassificationRank(int raceId) throws
IDNotRecognisedException {
536         Race race = getRaceIfValidElseThrow(raceId);
537         List<Stage> stageList = race.getStageList();
538         for (Stage stage : stageList) {
539             if (stage.hasNoResult()) {
540                 return new int[0];
541             }
542         }
543
544         Map<Integer, Integer> map = new HashMap<>();
545         for (Stage stage : race.getStageList()) {
546             LocalTime[] adjustedElapsedTimes = stage.getAdjustedElapsedTimes();
547             int[] allIds = stage.getAllIds();
548             for (int i = 0; i < allIds.length; i++) {
549                 int id = allIds[i];
550                 int time = adjustedElapsedTimes[i].toSecondOfDay();
551                 map.put(id, map.getOrDefault(id, 0) + time);
552             }
553         }
```

```
554
555         List<Integer> idList = new ArrayList<>(map.keySet());
556
557         idList.sort((a, b) -> {
558             Integer pointA = map.get(a);
559             Integer pointB = map.get(b);
560             return pointA.compareTo(pointB);
561         });
562
563         int[] ans = new int[idList.size()];
564         for (int i = 0; i < ans.length; i++) {
565             ans[i] = idList.get(i);
566         }
567         return ans;
568     }
569
570     public int[] getRidersPointClassificationRank(int raceId) throws
    IDNotRecognisedException {
571         Race race = getRaceIfValidElseThrow(raceId);
572         List<Stage> stageList = race.getStageList();
573         for (Stage stage : stageList) {
574             if (stage.hasNoResult()) {
575                 return new int[0];
576             }
577         }
578
579         Map<Integer, Integer> map = new HashMap<>();
580         for (Stage stage : race.getStageList()) {
581             int[] ridersPointsInStage = stage.getRidersPointsInStage();
582             int[] allIds = stage.getAllIds();
583             for (int i = 0; i < allIds.length; i++) {
584                 int id = allIds[i];
585                 map.put(id, map.getOrDefault(id, 0) + ridersPointsInStage[i]);
586             }
587         }
588
589         List<Integer> idList = new ArrayList<>(map.keySet());
590
591         idList.sort((a, b) -> {
592             Integer pointA = map.get(a);
593             Integer pointB = map.get(b);
594             return pointA.compareTo(pointB);
595         });
596
597         int[] ans = new int[idList.size()];
598         for (int i = 0; i < ans.length; i++) {
599             ans[i] = idList.get(i);
600         }
601         return ans;
602     }
603
604     public int[] getRidersMountainPointClassificationRank(int raceId) throws
    IDNotRecognisedException {
605         Race race = getRaceIfValidElseThrow(raceId);
606         List<Stage> stageList = race.getStageList();
607         for (Stage stage : stageList) {
608             if (stage.hasNoResult()) {
609                 return new int[0];
610             }
611         }
```

```
612
613         Map<Integer, Integer> map = new HashMap<>();
614         for (Stage stage : race.getStageList()) {
615             int[] ridersPointsInStage = stage.getRidersMountainPointsInStage();
616             int[] allIds = stage.getAllIds();
617             for (int i = 0; i < allIds.length; i++) {
618                 int id = allIds[i];
619                 map.put(id, map.getOrDefault(id, 0) + ridersPointsInStage[i]);
620             }
621         }
622
623         List<Integer> idList = new ArrayList<>(map.keySet());
624
625         idList.sort((a, b) -> {
626             Integer pointA = map.get(a);
627             Integer pointB = map.get(b);
628             return pointA.compareTo(pointB);
629         });
630
631         int[] ans = new int[idList.size()];
632         for (int i = 0; i < ans.length; i++) {
633             ans[i] = idList.get(i);
634         }
635         return ans;
636     }
637
638     //*********************************//
639     //        Our Private methods      //
640     //*********************************//
641
642     private Race getRaceIfValidElseThrow(String name) throws
    NameNotRecognisedException {
643         Race race = getRace(name);
644         if (race == null) {
645             throw new NameNotRecognisedException("name does not match to any race in
    the system.");
646         }
647         return race;
648     }
649
650     private Race getRace(int raceId) {
651         for (Race race : raceList) {
652             if (race.getId() == raceId) {
653                 return race;
654             }
655         }
656         return null;
657     }
658
659     private Race getRace(String name) {
660         for (Race race : raceList) {
661             if (race.getName().equals(name)) {
662                 return race;
663             }
664         }
665         return null;
666     }
667
668     private Race getRaceIfValidElseThrow(int raceId) throws IDNotRecognisedException
    {
```

```
669         Race race = getRace(raceId);
670         if (race == null) {
671             throw new IDNotRecognisedException(raceId + " does not exists.");
672         }
673         return race;
674     }
675 
676     private Stage getStageFromAnyRace(int stageId) throws IDNotRecognisedException {
677         for (Race race : raceList) {
678             Stage stage = race.getStage(stageId);
679             if (stage != null) {
680                 return stage;
681             }
682         }
683         throw new IDNotRecognisedException("ID does not match to any stage in the
    system.");
684     }
685 
686     private Stage getStageForSegmentIdFromAnyRace(int segmentId) throws
    IDNotRecognisedException {
687         for (Race race : raceList) {
688             for (Stage stage : race.getStageList()) {
689                 for (Segment segment : stage.getSegmentList()) {
690                     if (segment.getId() == segmentId) {
691                         return stage;
692                     }
693                 }
694             }
695         }
696         throw new IDNotRecognisedException("ID does not match to any segment in the
    system.");
697     }
698 
699     private Segment getSegmentFromAnyRace(int segmentId) throws
    IDNotRecognisedException {
700         for (Race race : raceList) {
701             for (Stage stage : race.getStageList()) {
702                 for (Segment segment : stage.getSegmentList()) {
703                     if (segment.getId() == segmentId) {
704                         return segment;
705                     }
706                 }
707             }
708         }
709         throw new IDNotRecognisedException("ID does not match to any segment in the
    system.");
710     }
711 
712     private Team getTeamIfValidElseThrow(int teamId) throws IDNotRecognisedException
    {
713         for (Team team : teamList) {
714             if (team.getId() == teamId) {
715                 return team;
716             }
717         }
718         throw new IDNotRecognisedException(teamId + " does not exists.");
719     }
720 
721     private Rider getRiderIfValidElseThrow(int riderId) throws
    IDNotRecognisedException {
```

```
722         for (Team team : teamList) {
723             for (Rider rider : team.getRiderList()) {
724                 if (rider.getId() == riderId) {
725                     return rider;
726                 }
727             }
728         }
729 
730         throw new IDNotRecognisedException("ID does not match to any rider in the
    system");
731     }
732 
733     private Team getTeamForRiderElseThrow(int riderId) throws
    IDNotRecognisedException {
734         for (Team team : teamList) {
735             for (Rider rider : team.getRiderList()) {
736                 if (rider.getId() == riderId) {
737                     return team;
738                 }
739             }
740         }
741 
742         throw new IDNotRecognisedException("ID does not match to any rider in the
    system");
743     }
744 
745     private void validateName(String name) throws InvalidNameException {
746         if (name == null || name.isEmpty() || name.length() > 30 || name.contains("
    ") || name.contains("\t")) {
747             throw new InvalidNameException(name + " is not valid.");
748         }
749     }
750 }
751 
```

```java
package cycling;

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

public class Race implements Serializable {
    public static int availableId = 1;

    private int id;
    private String name;
    private String description;

    private List<Stage> stageList;

    public Race(String name, String description) {
        this.id = availableId;
        availableId += 1;

        this.name = name;
        this.description = description;
        stageList = new ArrayList<>();
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public List<Stage> getStageList() {
        return stageList;
    }

    @Override
    public String toString() {
        /*Any formatted string containing the race ID, name, description, the
                number of stages, and the total length (i.e., the sum of all stages'
                length).*/

        double totalLength = 0;
        for (Stage stage : stageList) {
            totalLength += stage.getLength();
        }

```

```
60          return "Race{" +
61                  "id=" + id +
62                  ", name='" + name + '\'' +
63                  ", description='" + description + '\'' +
64                  ", number of stages=" + stageList.size() +
65                  ", total length=" + totalLength +
66                  '}';
67      }
68
69      public int[] getStageIds() {
70          int[] ids = new int[stageList.size()];
71          for (int i = 0; i < stageList.size(); i++) {
72              ids[i] = stageList.get(i).getId();
73          }
74          return ids;
75      }
76
77      public Stage getStage(int stageId) {
78          for (Stage stage : stageList) {
79              if (stage.getId() == stageId) {
80                  return stage;
81              }
82          }
83          return null;
84      }
85
86      public Stage getStage(String stageName) {
87          for (Stage stage : stageList) {
88              if (stage.getStageName().equals(stageName)) {
89                  return stage;
90              }
91          }
92          return null;
93      }
94
95      public void add(Stage stage) {
96          stageList.add(stage);
97      }
98
99      public void remove(Stage stage) {
100         stageList.remove(stage);
101     }
102 }
103
```

```java
package cycling;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Stage implements Serializable {
    public static int availableId = 1;

    private int id;
    private String stageName;
    private String description;
    private double length;
    private LocalDateTime startTime;
    private StageType stageType;
    private String stageState;

    private List<Segment> segmentList;
    private List<RiderResult> stageResult;

    public Stage(String stageName, String description, double length, LocalDateTime startTime, StageType stageType) {
        this.id = availableId;
        availableId += 1;

        this.stageName = stageName;
        this.description = description;
        this.length = length;
        this.startTime = startTime;
        this.stageType = stageType;
        this.stageState = "";

        this.segmentList = new ArrayList<>();
        this.stageResult = new ArrayList<>();
    }

    public int getId() {
        return id;
    }

    public double getLength() {
        return length;
    }

    public String getStageName() {
        return stageName;
    }

    public String getStageState() {
        return stageState;
    }

    public void setStageState(String stageState) {
        this.stageState = stageState;
    }
```

```
59
60      public StageType getStageType() {
61          return stageType;
62      }
63
64      public List<Segment> getSegmentList() {
65          return segmentList;
66      }
67
68      public void add(Segment segment) {
69          segmentList.add(segment);
70      }
71
72      public void remove(Segment segment) {
73          segmentList.remove(segment);
74      }
75
76      boolean hasResult(int riderId) {
77          for (int i = 0; i < stageResult.size(); i++) {
78              if (stageResult.get(i).getRiderId() == riderId) {
79                  return true;
80              }
81          }
82          return false;
83      }
84
85      void addResult(int stageId, int riderId, LocalTime[] localTimes) {
86          if (hasResult(riderId)) {
87              return;
88          }
89          RiderResult riderResult = new RiderResult(stageId, riderId, localTimes);
90          stageResult.add(riderResult);
91      }
92
93      RiderResult getResult(int riderId) {
94          for (int i = 0; i < stageResult.size(); i++) {
95              RiderResult riderResult = stageResult.get(i);
96              if (riderResult.getRiderId() == riderId) {
97                  return riderResult;
98              }
99          }
100         return null;
101     }
102
103     public void removeResult(RiderResult result) {
104         stageResult.remove(result);
105     }
106
107     public boolean hasNoResult() {
108         return stageResult.isEmpty();
109     }
110
111     public int[] getRiderRanks() {
112
113         stageResult.sort((a, b) -> {
114             if (a.getElapsedSeconds() < b.getElapsedSeconds()) return -1;
115             if (a.getElapsedSeconds() > b.getElapsedSeconds()) return +1;
116             return 0;
117         });
118
```

```
119         int[] ids = new int[stageResult.size()];
120         for (int i = 0; i < ids.length; i++) {
121             ids[i] = stageResult.get(i).getRiderId();
122         }
123         return ids;
124     }
125 
126     public int[] getAllIds() {
127         int[] ids = new int[stageResult.size()];
128         for (int i = 0; i < ids.length; i++) {
129             ids[i] = stageResult.get(i).getRiderId();
130         }
131         return ids;
132     }
133 
134     public LocalTime[] getAdjustedElapsedTimes() {
135         sortResultsByFinishTime();
136 
137         LocalTime[] localTimes = new LocalTime[stageResult.size()];
138         localTimes[0] = stageResult.get(0).getFinishTime();
139         for (int i = 1; i < localTimes.length; i++) {
140             int i_1_second = stageResult.get(i - 1).getFinishTime().toSecondOfDay();
141             int i_second = stageResult.get(i).getFinishTime().toSecondOfDay();
142             if (i_1_second + 1 == i_second) {
143                 localTimes[i] = localTimes[i - 1];
144             } else {
145                 localTimes[i] = stageResult.get(i).getFinishTime();
146             }
147         }
148 
149         return localTimes;
150     }
151 
152     private void sortResultsByFinishTime() {
153         stageResult.sort((a, b) -> {
154             int secondA = a.getFinishTime().toSecondOfDay();
155             int secondB = b.getFinishTime().toSecondOfDay();
156             if (secondA < secondB) return -1;
157             if (secondA > secondB) return +1;
158             return 0;
159         });
160     }
161 
162     public LocalTime getAdjustedElapsedTime(int riderId) {
163         // we may think that all start time is same
164         // finish - start is same as comparing with finish
165         // but let's try with finish time first as suggested
166         sortResultsByFinishTime();
167 
168         int i = 0;
169         for (; i < stageResult.size(); i++) {
170             if (stageResult.get(i).getRiderId() == riderId) {
171                 break;
172             }
173         }
174 
175         if (i == stageResult.size()) {
176             return null;
177         }
178 
```

```
179         //        i
180         //1 3 4 5 9
181         //0 1 2 3 4
182         LocalTime localTime = null;
183         for (int j = i; j >= 1; j--) {
184             int aj = stageResult.get(j).getFinishTime().toSecondOfDay();
185             int aj_1 = stageResult.get(j - 1).getFinishTime().toSecondOfDay();
186             if (aj_1 + 1 == aj) {
187                 continue;
188             }
189             localTime = stageResult.get(j).getFinishTime();
190             break;
191         }
192         return localTime;
193     }
194 
195     public List<RiderResult> getStageResult() {
196         return stageResult;
197     }
198 
199     public int[] getRidersPointsInStage() {
200 
201         Map<Integer, Integer> riderPoints = new HashMap<>();
202         // init points to 0
203         for (RiderResult result : stageResult) {
204             riderPoints.put(result.getRiderId(), 0);
205         }
206 
207         // for first 15 riders if any
208         var pointsForIntermediateSprint = new int[]{20, 17, 15, 13, 11, 10, 9, 8, 7,
    6, 5, 4, 3, 2, 1};
209         int totalRidersToGivePoints = Math.min(15, stageResult.size());
210 
211         for (int i = 0; i < segmentList.size(); i++) {
212             Segment segment = segmentList.get(i);
213             SegmentType segmentType = segment.getSegmentType();
214             if (segmentType != SegmentType.SPRINT) {
215                 // ignore if not an intermediate sprint.
216                 continue;
217             }
218 
219             // for segment i sort the stageResult first based on
    segmentTimes.get(i).toSecondsOfDay()
220             sortBySegmentIndex(i);
221 
222             for (int j = 0; j < totalRidersToGivePoints; j++) {
223                 RiderResult riderResult = stageResult.get(j);
224                 int riderId = riderResult.getRiderId();
225                 riderPoints.put(riderId, riderPoints.get(riderId) +
    pointsForIntermediateSprint[j]);
226             }
227         }
228 
229         // now allocate points based on who finishes early
230         stageResult.sort((a, b) -> {
231             if (a.getElapsedSeconds() < b.getElapsedSeconds()) return -1;
232             if (a.getElapsedSeconds() > b.getElapsedSeconds()) return +1;
233             return 0;
234         });
235 
```

```
236         Map<StageType, int[]> points = new HashMap<>();
237         points.put(StageType.FLAT, new int[]{50, 30, 20, 18, 16, 14, 12, 10, 8, 7,
    6, 5, 4, 3, 2});
238         points.put(StageType.MEDIUM_MOUNTAIN, new int[]{30, 25, 22, 19, 17, 15, 13,
    11, 9, 7, 6, 5, 4, 3, 2});
239         points.put(StageType.HIGH_MOUNTAIN, new int[]{20, 17, 15, 13, 11, 10, 9, 8,
    7, 6, 5, 4, 3, 2, 1});
240         points.put(StageType.TT, new int[]{20, 17, 15, 13, 11, 10, 9, 8, 7, 6, 5, 4,
    3, 2, 1});
241 
242         // for first 15 riders if any
243         int[] pointsForThisStageType = points.get(stageType);
244 
245         for (int j = 0; j < totalRidersToGivePoints; j++) {
246             RiderResult riderResult = stageResult.get(j);
247             int riderId = riderResult.getRiderId();
248             riderPoints.put(riderId, riderPoints.get(riderId) +
    pointsForThisStageType[j]);
249         }
250 
251         int[] ans = new int[stageResult.size()];
252         for (int i = 0; i < ans.length; i++) {
253             ans[i] = riderPoints.get(stageResult.get(i).getRiderId());
254         }
255 
256         return ans;
257     }
258 
259     private void sortBySegmentIndex(int segmentIndexToUse) {
260         stageResult.sort((a, b) -> {
261 
262             LocalTime[] segmentTimesOfA = a.getSegmentTimes();
263             LocalTime[] segmentTimesOfB = b.getSegmentTimes();
264 
265             LocalTime localTimeOfA = segmentTimesOfA[segmentIndexToUse];
266             LocalTime localTimeOfB = segmentTimesOfB[segmentIndexToUse];
267 
268             // 2 AM == 2 * 3600 sec
269             int secondA = localTimeOfA.toSecondOfDay();
270             int secondB = localTimeOfB.toSecondOfDay();
271 
272             if (secondA < secondB) return -1;
273             if (secondA > secondB) return +1;
274 
275             return 0;
276         });
277     }
278 
279     public int[] getRidersMountainPointsInStage() {
280 
281         Map<SegmentType, int[]> points = new HashMap<>();
282         points.put(SegmentType.HC, new int[]{20, 15, 12, 10, 8, 6, 4, 2});
283         points.put(SegmentType.C1, new int[]{10, 8, 6, 4, 2, 1, 0, 0});
284         points.put(SegmentType.C2, new int[]{5, 3, 2, 1, 0, 0, 0, 0});
285         points.put(SegmentType.C3, new int[]{2, 1, 0, 0, 0, 0, 0, 0});
286         points.put(SegmentType.C4, new int[]{1, 0, 0, 0, 0, 0, 0, 0});
287 
288         Map<Integer, Integer> riderPoints = new HashMap<>();
289         // init points to 0
290         for (RiderResult result : stageResult) {
```

```
291             riderPoints.put(result.getRiderId(), 0);
292         }
293 
294         // for first 8 riders if any
295         int totalRidersToGivePoints = Math.min(8, stageResult.size());
296 
297         for (int i = 0; i < segmentList.size(); i++) {
298             Segment segment = segmentList.get(i);
299             SegmentType segmentType = segment.getSegmentType();
300             if (segmentType == SegmentType.SPRINT) {
301                 // ignore if an intermediate sprint.
302                 continue;
303             }
304 
305             int[] pointsToGive = points.get(segmentType);
306 
307             // for segment i sort the stageResult first based on
segmentTimes.get(i).toSecondsOfDay()
308             sortBySegmentIndex(i);
309 
310             for (int j = 0; j < totalRidersToGivePoints; j++) {
311                 RiderResult riderResult = stageResult.get(j);
312                 int riderId = riderResult.getRiderId();
313                 riderPoints.put(riderId, riderPoints.get(riderId) +
pointsToGive[j]);
314             }
315         }
316 
317         // now allocate points based on who finishes early
318         sortResultsByFinishTime();
319 
320         int[] ans = new int[stageResult.size()];
321         for (int i = 0; i < ans.length; i++) {
322             ans[i] = riderPoints.get(stageResult.get(i).getRiderId());
323         }
324 
325         return ans;
326     }
327 }
328 
```

```java
package cycling;

import java.io.Serializable;

public class Segment implements Serializable {
    public static int availableId = 1;

    private int id;
    private double location;
    private SegmentType segmentType;
    private double averageGradient;
    private double length;

    public Segment(double location, SegmentType segmentType, double averageGradient, double length) {
        this(location, segmentType);
        this.averageGradient = averageGradient;
        this.length = length;
    }

    public Segment(double location, SegmentType segmentType) {
        this.id = availableId;
        availableId += 1;
        this.location = location;
        this.segmentType = segmentType;
    }

    public int getId() {
        return id;
    }

    public double getLocation() {
        return location;
    }

    public SegmentType getSegmentType() {
        return segmentType;
    }
}
```

```java
package cycling;

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

public class Team implements Serializable {
    public static int availableId = 1;

    private int id;
    private String name;
    private String description;

    private List<Rider> riderList;

    public Team(String name, String description) {
        this.id = availableId;
        availableId += 1;

        this.name = name;
        this.description = description;
        riderList = new ArrayList<>();
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public List<Rider> getRiderList() {
        return riderList;
    }

    public void add(Rider rider) {
        riderList.add(rider);
    }

    public void remove(Rider rider) {
        riderList.remove(rider);
    }
}

```

```java
package cycling;

import java.io.Serializable;

public class Rider implements Serializable {
    public static int availableId = 1;

    private int id;
    private String name;
    private int yearOfBirth;

    public Rider(String name, int yearOfBirth) {
        this.id = availableId;
        availableId += 1;

        this.name = name;
        this.yearOfBirth = yearOfBirth;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }
}

```

```java
package cycling;

import java.io.Serializable;
import java.time.Duration;
import java.time.LocalTime;

public class RiderResult implements Serializable {
    private int stageId;
    private int riderId;
    private LocalTime[] segmentTimes;
    private LocalTime startTime, finishTime;
    private int elapsedSeconds;

    public RiderResult(int stageId, int riderId, LocalTime[] localTimes) {
        this.stageId = stageId;
        this.riderId = riderId;

        this.segmentTimes = new LocalTime[localTimes.length - 1];
        for (int i = 1; i < localTimes.length - 1; i++) {
            segmentTimes[i - 1] = localTimes[i];
        }

        this.startTime = localTimes[0];
        this.finishTime = localTimes[localTimes.length - 1];
        this.elapsedSeconds = (int) Math.abs(Duration.between(startTime, finishTime).toSeconds());

        int hour = elapsedSeconds / 3600;
        int minute = (elapsedSeconds - hour * 3600) / 60;
        int second = elapsedSeconds - hour * 3600 - minute * 60;
        segmentTimes[segmentTimes.length - 1] = LocalTime.of(hour, minute, second);
    }

    public int getStageId() {
        return stageId;
    }

    public int getRiderId() {
        return riderId;
    }

    public LocalTime[] getSegmentTimes() {
        return segmentTimes;
    }

    public LocalTime getStartTime() {
        return startTime;
    }

    public LocalTime getFinishTime() {
        return finishTime;
    }

    public int getElapsedSeconds() {
        return elapsedSeconds;
    }
}

```

```java
package cycling;

import java.io.Serializable;
import java.util.List;

public class DataToSerializeDeserialize implements Serializable {
    public int availableRaceId;
    public int availableRiderId;
    public int availableSegmentId;
    public int availableStageId;
    public int availableTeamId;

    public List<Race> raceList;
    public List<Team> teamList;
}

```