1 CyclingPortal.java

```
1  package cycling;
2
3  import java.io.FileInputStream;
4  import java.io.FileOutputStream;
5  import java.io.IOException;
6  import java.io.ObjectInputStream;
7  import java.io.ObjectOutputStream;
8  import java.time.Duration;
9  import java.time.LocalDateTime;
10 import java.time.LocalTime;
11 import java.util.ArrayList;
12 import java.util.Arrays;
13 import java.util.Collections;
14 import java.util.Hashtable;
15 import java.util.Set;
16
17
18 /**
19  * CyclingPortal is a minimally compiling, but non-functioning implementor
20  * of the CyclingPortalInterface interface.
21  *
22  * @author Diogo Pacheco
23  * @version 1.0
24  *
25  */
26 public class CyclingPortal implements CyclingPortalInterface {
27    ArrayList<Race> arrayListOfRaces = new ArrayList<>();
28    ArrayList<Team> arrayListOfTeams = new ArrayList<>();
29
30    @Override
31    public int[] getRaceIds() {
32       if (arrayListOfRaces.size() == 0) {
33          return new int[0];
34       } else {
35          ArrayList<Integer> arrayListOfRaceIDs = new ArrayList<>();
36          for (Race race:arrayListOfRaces) {
37             arrayListOfRaceIDs.add(race.getRaceId());
38       }
39       int[] arrayNew = new int[arrayListOfRaceIDs.size()];
40       for (int i=0; i < arrayNew.length; i++) {
41          arrayNew[i] = arrayListOfRaceIDs.get(i).intValue();
42       }
43       return arrayNew;
44       }
45    }
46
47    @Override
48    public int createRace(String name, String description) throws IllegalNameException, InvalidNameException
          {
49       Race newRace = new Race(name, description);
50       for (Race race:arrayListOfRaces){
51          if (race.getName().equals(name)){
```

```
52              throw new IllegalNameException("The race name already exists in the platform");
53          }
54          if (name == null || name.isEmpty() || name.length() == 0 || name.length() > 30 || name.contains("
                ")) {
55              throw new InvalidNameException("Race name entered can't be empty, have more than 30 characters,
                    or include spaces");
56          }
57      }
58      arrayListOfRaces.add(newRace);
59      return newRace.getRaceId();
60  }
61
62  @Override
63  public String viewRaceDetails(int raceId) throws IDNotRecognisedException {
64      for (Race race:arrayListOfRaces){
65          if (race.getRaceId()==(raceId)){
66              String concatenatedString = String.valueOf(race.getRaceId()) + " " + race.getName()
67              + " " + race.getDescription() +" "+ String.valueOf(race.getNumberOfStages())
68              + " "+ String.valueOf(race.getTotalLength());
69              return concatenatedString;
70          }
71      }
72      throw new IDNotRecognisedException("The ID entered does not match to any race in the system");
73  }
74
75  @Override
76  public void removeRaceById(int raceId) throws IDNotRecognisedException {
77      int initialLength = arrayListOfRaces.size();
78      for (Race race:arrayListOfRaces ){
79          if (race.getRaceId() == (raceId)) {
80              arrayListOfRaces.remove(race);
81              break;
82          }
83      }
84      if (arrayListOfRaces.size() == initialLength) {
85          throw new IDNotRecognisedException("The ID entered does not match to any race in the system");
86      }
87  }
88
89  @Override
90  public int getNumberOfStages(int raceId) throws IDNotRecognisedException {
91      for (Race race:arrayListOfRaces){
92          if (race.getRaceId()==(raceId)){
93              int numberOfStages = race.getNumberOfStages();
94              return numberOfStages;
95          }
96      }throw new IDNotRecognisedException("The ID entered does not match to any race in the system");
97  }
98
99  @Override
100 public int addStageToRace(int raceId, String stageName, String description, double length, LocalDateTime
        startTime, StageType type)
101         throws IDNotRecognisedException, IllegalNameException, InvalidNameException, InvalidLengthException
                {
102     for (Race race:arrayListOfRaces){
```

```
103         if (race.getRaceId()==(raceId)){
104            for (Stage s:race.arrayListOfStages){
105               if (s.getStageName().equals(stageName)){
106                  throw new IllegalNameException("The stage name already exists in the platform");
107               }
108            }
109            if (stageName==null || stageName.isEmpty()|| stageName.length()>30|| stageName.contains(" ")){
110               throw new InvalidNameException("Name entered can't be empty, have more than 30 characters, or
                      include spaces");
111            }
112            if (length<5){
113               throw new InvalidLengthException("The length of the stage must be longer than 5km");
114            }
115            int stageId =race.createStage(raceId, stageName, description, length, startTime, type);
116            return stageId;
117         }
118      }throw new IDNotRecognisedException("The ID entered does not match to any race in the system");
119   }
120
121   @Override
122   public int[] getRaceStages(int raceId) throws IDNotRecognisedException {
123      for (Race race:arrayListOfRaces){
124         if (race.getRaceId()==(raceId)){
125            ArrayList<Integer> arrayListOfStageIDs = race.getRaceStages();
126            //int[] array = arrayListOfStageIDs.toArray(new int[arrayListOfStageIDs.size()]);
127            int[] arrayNew = new int[arrayListOfStageIDs.size()];
128            for (int i=0; i < arrayNew.length; i++) {
129               arrayNew[i] = arrayListOfStageIDs.get(i).intValue();
130            }
131            return arrayNew;
132         }
133      }throw new IDNotRecognisedException("The ID entered does not match to any race in the system");
134   }
135
136   @Override
137   public double getStageLength(int stageId) throws IDNotRecognisedException {
138      for (Race race: arrayListOfRaces){
139         for (Stage s:race.arrayListOfStages){
140            if (s.getStageId()==stageId){
141               double stageLength = s.getStageLength();
142               return stageLength;
143            }
144         }
145      }throw new IDNotRecognisedException("The ID entered does not match to any stage in the system");
146   }
147
148   @Override
149   public void removeStageById(int stageId) throws IDNotRecognisedException {
150      int initialLength=0;
151      Race raceInQuestion = null;
152      for (Race race: arrayListOfRaces){
153         initialLength = race.arrayListOfStages.size();
154         race.removeStage(stageId);
155         raceInQuestion = race;
156         break;
```

```
157         }if (raceInQuestion.arrayListOfStages.size()==initialLength){
158             throw new IDNotRecognisedException("The ID entered does not match to any stage in the system");
159         }
160     }
161 
162     @Override
163     public int addCategorizedClimbToStage(int stageId, Double location, SegmentType type, Double
            averageGradient,
164             Double length) throws IDNotRecognisedException, InvalidLocationException,
                    InvalidStageStateException,
165             InvalidStageTypeException {
166         for (Race race: arrayListOfRaces){
167             for (Stage s:race.arrayListOfStages){
168                 if (s.getStageId()==stageId){
169                     if (location>s.getStageLength()){
170                         throw new InvalidLocationException("The entered finish location is not within the stage
                                length");
171                     }
172                     if (s.getStageState().equals("waiting for results")){
173                         throw new InvalidStageStateException("Still waiting for results");
174                     }
175                     if (s.getStageType().equals(StageType.TT)){
176                         throw new InvalidStageTypeException("Time trial stages cannot contain any segments");
177                     }
178                     int climbId = s.addCategorizedClimbToStage(stageId, location, type, averageGradient, length);
179                     return climbId;
180                 }
181             }
182         }throw new IDNotRecognisedException("The ID entered does not match to any stage in the system");
183     }
184 
185     @Override
186     public int addIntermediateSprintToStage(int stageId, double location) throws IDNotRecognisedException,
187             InvalidLocationException, InvalidStageStateException, InvalidStageTypeException {
188         for (Race race: arrayListOfRaces){
189             for (Stage s:race.arrayListOfStages){
190                 if (s.getStageId()==stageId){
191                     if (location>s.getStageLength()){
192                         throw new InvalidLocationException("The entered finish location is not within the stage
                                length");
193                     }
194                     if (s.getStageState().equals("waiting for results")){
195                         throw new InvalidStageStateException("Still waiting for results");
196                     }
197                     if (s.getStageType().equals(StageType.TT)){
198                         throw new InvalidStageTypeException("Time trial stages cannot contain any segments");
199                     }
200                     int sprintId = s.addIntermediateSprintToStage(stageId, location);
201                     return sprintId;
202                 }
203             }
204         }throw new IDNotRecognisedException("The ID entered does not match to any stage in the system");
205     }
206 
207     @Override
```

```
208     public void removeSegment(int segmentId) throws IDNotRecognisedException, InvalidStageStateException {
209        int initialLength =0;
210        Stage stageInQuestion = null;
211        outerloop:
212        for (Race race: arrayListOfRaces){
213           for (Stage i:race.arrayListOfStages){
214              if (i.getStageState().equals("waiting for results")){
215                 throw new InvalidStageStateException("Still waiting for results");
216              }
217              initialLength= i.arrayListOfSegments.size();
218              i.removeSegment(segmentId);
219              stageInQuestion = i;
220              break outerloop;
221           }
222        }if (stageInQuestion.arrayListOfSegments.size()==initialLength){
223           throw new IDNotRecognisedException("The ID entered does not match to any segment in the system");
224        }
225     }
226
227     @Override
228     public void concludeStagePreparation(int stageId) throws IDNotRecognisedException,
           InvalidStageStateException {
229        int flag = 0;
230        for (Race race: arrayListOfRaces){
231           for (Stage s:race.arrayListOfStages){
232              if (s.getStageId()==stageId){
233                 if (s.getStageState().equals("waiting for results")){
234                    throw new InvalidStageStateException("Still waiting for results");
235                 }
236                 s.concludeStagePreparation();
237                 flag=1;
238              }
239           }
240        }if (flag ==0 ){
241           throw new IDNotRecognisedException("The ID entered does not match to any stage in the system");
242        }
243     }
244
245     @Override
246     public int[] getStageSegments(int stageId) throws IDNotRecognisedException {
247        for (Race race: arrayListOfRaces){
248           for (Stage s:race.arrayListOfStages){
249              if (s.getStageId()==stageId){
250                 ArrayList<Integer> arrayListOfSegmentIDs = s.getStageSegments();
251                 int[] arrayNew = new int[arrayListOfSegmentIDs.size()];
252                 for (int i=0; i < arrayNew.length; i++) {
253                    arrayNew[i] = arrayListOfSegmentIDs.get(i).intValue();
254                 }
255                 return arrayNew;
256              }
257           }
258        }throw new IDNotRecognisedException("The ID entered does not match to any stage in the system");
259     }
260
261     @Override
```

```
262     public int createTeam(String name, String description) throws IllegalNameException, InvalidNameException
            {
263         for (Team team:arrayListOfTeams) {
264             if (team.getName().equals(name)) {
265                 throw new IllegalNameException("The team name already exists in the platform");
266             }
267             if (name == null || name.isEmpty() || name.length() == 0 || name.length() > 30 || name.contains("
                    ")) {
268                 throw new InvalidNameException("Team name entered can't be empty, have more than 30 characters,
                        or include spaces");
269             }
270         }
271         Team newTeam = new Team(name, description);
272         arrayListOfTeams.add(newTeam);
273         return newTeam.getTeamId();
274     }
275
276     @Override
277     public void removeTeam(int teamId) throws IDNotRecognisedException {
278         int initialLength = arrayListOfTeams.size();
279         for (Team team:arrayListOfTeams){
280             if (team.getTeamId() == (teamId)){
281                 arrayListOfTeams.remove(team);
282                 break;
283             }
284         }
285         if (arrayListOfTeams.size() == initialLength){
286             throw new IDNotRecognisedException("The ID entered does not match to any team in the system");
287         }
288     }
289
290     @Override
291     public int[] getTeams() {
292         if (arrayListOfTeams.size() == 0) {
293             return new int[0];
294         } else {
295         ArrayList<Integer> arrayListOfTeamIDs = new ArrayList<>();
296         for (Team team:arrayListOfTeams){
297             arrayListOfTeamIDs.add(team.getTeamId());
298         }
299         Integer[] array = arrayListOfTeamIDs.toArray(new Integer[arrayListOfTeamIDs.size()]);
300         int[] arrayNew = new int[array.length];
301         Arrays.setAll(arrayNew, i -> array[i]);
302         return arrayNew;
303         }
304     }
305
306     @Override
307     public int[] getTeamRiders(int teamId) throws IDNotRecognisedException {
308         for (Team team:arrayListOfTeams){
309             if (team.getTeamId() == (teamId)){
310                 ArrayList<Integer> arrayListOfRiderIDs = team.getTeamRiders();
311                 Integer[] array = arrayListOfRiderIDs.toArray(new Integer[arrayListOfRiderIDs.size()]);
312                 int[] arrayNew = new int[array.length];
313                 Arrays.setAll(arrayNew, i -> array[i]);
```

```
314             return arrayNew;
315          }
316       }
317       throw new IDNotRecognisedException("The ID entered does not match to any team in the system");
318    }
319
320    @Override
321    public int createRider(int teamID, String name, int yearOfBirth)
322          throws IDNotRecognisedException, IllegalArgumentException {
323       for (Team team:arrayListOfTeams){
324          if (team.getTeamId() == (teamID)){
325             if (name == null || yearOfBirth < 1900){
326                throw new IllegalArgumentException("Name of rider cannot be empty and year of birth cannot be
                       less than 1900");
327             }
328             int riderID = team.createRider(teamID, name, yearOfBirth);
329             return riderID;
330          }
331       }
332       throw new IDNotRecognisedException("The ID entered does not match to any team in the system");
333    }
334
335    @Override
336    public void removeRider(int riderId) throws IDNotRecognisedException {
337       int initialLength=0;
338       Team teamInQuestion = null;
339       outerloop:
340       for (Team team: arrayListOfTeams){
341          //initialLength = team.arrayListOfRiders.size();
342          for (Rider rider: team.arrayListOfRiders){
343             if (rider.getRiderId()==riderId){
344                initialLength = team.arrayListOfRiders.size();
345                team.removeRider(riderId);
346                teamInQuestion = team;
347                break outerloop;
348             }
349          }
350       }for (Race race: arrayListOfRaces){
351          for (Stage s:race.arrayListOfStages){
352             for (Result result: s.arrayListOfResults){
353                if (result.getRiderId()==riderId){
354                   s.arrayListOfResults.remove(result);
355                }
356             }
357          }
358       }if (teamInQuestion.arrayListOfRiders.size() == initialLength){
359          throw new IDNotRecognisedException("The ID entered does not match to any rider in the system");
360       }
361    }
362
363    @Override
364    public void registerRiderResultsInStage(int stageId, int riderId, LocalTime... checkpoints)
365          throws IDNotRecognisedException, DuplicatedResultException, InvalidCheckpointsException,
366          InvalidStageStateException {
367       int initialLength=0;
```

```
368         Stage stageInQuestion = null;
369         outerloop:
370         for (Race race: arrayListOfRaces){
371            for (Stage s:race.arrayListOfStages){
372               if (s.getStageId()==stageId){
373                  initialLength = s.arrayListOfResults.size();
374                  for (Result result: s.arrayListOfResults){
375                     if (result.getRiderId()==riderId){
376                        throw new DuplicatedResultException("Rider already has a result for that stage");
377                     }
378                  }
379                  if (checkpoints.length != (s.arrayListOfSegments.size()+2)){
380                     throw new InvalidCheckpointsException("The number of checkpoints in the stage is invaliid");
381                  }
382                  if (!s.getStageState().equals("waiting for results")){
383                     throw new InvalidStageStateException("Results can only be added to a stage while it is
                           waiting for results");
384                  }
385                  s.registerRiderResultsInStage(stageId, riderId, checkpoints);
386                  stageInQuestion = s;
387                  break outerloop;
388               }
389            }
390         }if (stageInQuestion==null){
391            throw new IDNotRecognisedException("The ID entered does not match to either any rider or stage in
                  the system");
392         }
393         if (stageInQuestion.arrayListOfResults.size()==initialLength){
394            throw new IDNotRecognisedException("The ID entered does not match to either any rider or stage in
                  the system");
395         }
396      }
397
398      @Override
399      public LocalTime[] getRiderResultsInStage(int stageId, int riderId) throws IDNotRecognisedException {
400         for (Race race: arrayListOfRaces){
401            for (Stage s:race.arrayListOfStages){
402               if (s.getStageId()==stageId){
403                  ArrayList<LocalTime> arrayListOfRiderResults = s.getRiderResultsInStage(riderId);
404                  LocalTime[] array = arrayListOfRiderResults.toArray(new
                        LocalTime[arrayListOfRiderResults.size()]);
405                  if (array.length ==0){
406                     throw new IDNotRecognisedException("The ID entered does not match to either any rider or
                           stage in the system");
407                  }
408                  return array;
409               }
410            }
411         }throw new IDNotRecognisedException("The ID entered does not match to either any rider or stage in the
               system");
412      }
413
414      @Override
415      public LocalTime getRiderAdjustedElapsedTimeInStage(int stageId, int riderId) throws
            IDNotRecognisedException {
```

```
416         for (Race race: arrayListOfRaces){
417             for (Stage s:race.arrayListOfStages){
418                 if (s.getStageId()==stageId){
419                     LocalTime adjustedTime = s.getRiderAdjustedElapsedTimeInStage(riderId);
420                     if (adjustedTime == null){
421                         throw new IDNotRecognisedException("The ID entered does not match to either any rider or
                                stage in the system");
422                     }
423                     return adjustedTime;
424                 }
425             }
426         }throw new IDNotRecognisedException("The ID entered does not match to either any rider or stage in the
              system");
427     }
428 
429     @Override
430     public void deleteRiderResultsInStage(int stageId, int riderId) throws IDNotRecognisedException {
431         int initialLength = 0;
432         Stage stageInQuestion=null;
433         outerloop:
434         for (Race race: arrayListOfRaces){
435             for (Stage s:race.arrayListOfStages){
436                 if (s.getStageId()==stageId){
437                     initialLength = s.arrayListOfResults.size();
438                     s.deleteRiderResultsInStage(riderId);
439                     stageInQuestion = s;
440                     break outerloop;
441                 }
442             }
443         }if (stageInQuestion==null){
444             throw new IDNotRecognisedException("The ID entered does not match to either any rider or stage in
                    the system");
445         }
446         if (stageInQuestion.arrayListOfResults.size()==initialLength){
447             throw new IDNotRecognisedException("The ID entered does not match to any rider or stage in the
                    system");
448         }
449     }
450 
451     @Override
452     public int[] getRidersRankInStage(int stageId) throws IDNotRecognisedException {
453         ArrayList<Rider> allRiders = new ArrayList<>();
454         for (Team team:arrayListOfTeams){
455             for (Rider rider: team.arrayListOfRiders){
456                 allRiders.add(rider);
457             }
458         }
459 
460         for (Race race: arrayListOfRaces){
461             for (Stage s:race.arrayListOfStages){
462                 if (s.getStageId()==stageId){
463                     ArrayList<Integer> ridersRankInStage = s.getRidersRankInStage(allRiders);
464                     int[] arrayNew = new int[ridersRankInStage.size()];
465                     for (int i=0; i < arrayNew.length; i++) {
466                         arrayNew[i] = ridersRankInStage.get(i).intValue();
```

```
467                 }
468                 return arrayNew;
469             }
470         }
471     }throw new IDNotRecognisedException("The ID entered does not match to any stage in the system");
472 
473 }
474 
475 @Override
476 public LocalTime[] getRankedAdjustedElapsedTimesInStage(int stageId) throws IDNotRecognisedException {
477     ArrayList<Rider> allRiders = new ArrayList<>();
478     for (Team team:arrayListOfTeams){
479         for (Rider rider: team.arrayListOfRiders){
480             allRiders.add(rider);
481         }
482     }
483 
484     for (Race race: arrayListOfRaces){
485         for (Stage s:race.arrayListOfStages){
486             if (s.getStageId()==stageId){
487                 ArrayList<LocalTime> rankedAdjustedElapsedTimes =
                        s.getRankedAdjustedElapsedTimesInStage(allRiders);
488                 LocalTime[] arrayNew = new LocalTime[rankedAdjustedElapsedTimes.size()];
489                 for (int i=0; i < arrayNew.length; i++) {
490                     arrayNew[i] = rankedAdjustedElapsedTimes.get(i);
491                 }
492                 return arrayNew;
493             }
494         }
495     }throw new IDNotRecognisedException("The ID entered does not match to any stage in the system");
496 }
497 @Override
498 public int[] getRidersPointsInStage(int stageId) throws IDNotRecognisedException {
499     ArrayList<Rider> allRiders = new ArrayList<>();
500     for (Team team:arrayListOfTeams){
501         for (Rider rider: team.arrayListOfRiders){
502             allRiders.add(rider);
503         }
504     }
505 
506     for (Race race: arrayListOfRaces){
507         for (Stage s:race.arrayListOfStages){
508             if (s.getStageId()==stageId){
509                 ArrayList<Integer> ridersPointsInStage = s.getRidersPointsInStage(allRiders);
510                 int[] arrayNew = new int[ridersPointsInStage.size()];
511                 for (int i=0; i < arrayNew.length; i++) {
512                     arrayNew[i] = ridersPointsInStage.get(i).intValue();
513                 }
514                 return arrayNew;
515             }
516         }
517     }throw new IDNotRecognisedException("The ID entered does not match to any stage in the system");
518 }
519 
520 @Override
```

```
521     public int[] getRidersMountainPointsInStage(int stageId) throws IDNotRecognisedException {
522        ArrayList<Rider> allRiders = new ArrayList<>();
523        for (Team team:arrayListOfTeams){
524           for (Rider rider: team.arrayListOfRiders){
525              allRiders.add(rider);
526           }
527        }
528
529        for (Race race: arrayListOfRaces){
530           for (Stage s:race.arrayListOfStages){
531              if (s.getStageId()==stageId){
532                 if (s.getStageType().equals(StageType.TT)){
533                    return new int[allRiders.size()];
534                 }
535                 ArrayList<Integer> ridersMountainPointsInStage = s.getRidersMountainPointsInStage(allRiders);
536                 int[] arrayNew = new int[ridersMountainPointsInStage.size()];
537                 for (int i=0; i < arrayNew.length; i++) {
538                    arrayNew[i] = ridersMountainPointsInStage.get(i).intValue();
539                 }
540                 return arrayNew;
541              }
542           }
543        }throw new IDNotRecognisedException("The ID entered does not match to any stage in the system");
544     }
545
546     @Override
547     public void eraseCyclingPortal() {
548        arrayListOfRaces.clear();
549          arrayListOfTeams.clear();
550
551          Team.teamCounter = 0;
552        Race.raceCounter = 0;
553        Rider.riderCounter = 0;
554        Segment.segmentCounter = 0;
555          Stage.stageCounter = 0;
556     }
557
558     @Override
559     public void saveCyclingPortal(String filename) throws IOException {
560        ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(filename));
561          outputStream.writeObject(this);
562          outputStream.close();
563     }
564
565     @Override
566     public void loadCyclingPortal(String filename) throws IOException, ClassNotFoundException {
567        eraseCyclingPortal();
568        ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(filename));
569        Object portalObject = inputStream.readObject();
570        CyclingPortal newPortal = (CyclingPortal)portalObject;
571        this.arrayListOfTeams = newPortal.arrayListOfTeams;
572        this.arrayListOfRaces = newPortal.arrayListOfRaces;
573        inputStream.close();
574     }
575
```

```
576     @Override
577     public void removeRaceByName(String name) throws NameNotRecognisedException {
578        int initialLength = arrayListOfRaces.size();
579        for (Race race:arrayListOfRaces){
580           if ((race.getName()).equals(name)){
581              arrayListOfRaces.remove(race);
582              break;
583           }
584        }
585        if (arrayListOfRaces.size() == initialLength){
586           throw new NameNotRecognisedException("The name entered does not match to any race in the system");
587        }
588     }
589
590     @Override
591     public LocalTime[] getGeneralClassificationTimesInRace(int raceId) throws IDNotRecognisedException {
592        ArrayList<Integer> arrayListOfRaceIDs = new ArrayList<>();
593        for (Race race: arrayListOfRaces){
594           arrayListOfRaceIDs.add(race.getRaceId());
595        }
596        if (!arrayListOfRaceIDs.contains(raceId)){
597           throw new IDNotRecognisedException("The ID entered does not match to any race in the system");
598        }
599        int numberOfRiders = 0;
600          Hashtable<String, LocalTime> riderTotalAdjustedElapsedTimeDictionary = new Hashtable<String,
               LocalTime>();
601        for (Race race: arrayListOfRaces){
602             if (race.getRaceId()==(raceId)){
603                 if (race.getNumberOfStages()!=(0)){
604                     numberOfRiders = (race.getStages().get(0)).getNumberOfResults();
605                     for (Result result:((race.getStages().get(0)).getArrayListOfResults())){
606                         if (!riderTotalAdjustedElapsedTimeDictionary.containsKey("Rider
                                "+String.valueOf(result.getRiderId()))){
607                            riderTotalAdjustedElapsedTimeDictionary.put("Rider
                                    "+String.valueOf(result.getRiderId()),LocalTime.of(0,0,0,0));
608                         }
609                     }
610                 }
611             }
612          }
613          if (numberOfRiders==0){
614              return new LocalTime[0];
615          }
616          Set<String> setOfKeys = riderTotalAdjustedElapsedTimeDictionary.keySet();
617          if (numberOfRiders==riderTotalAdjustedElapsedTimeDictionary.size()){
618           for (Race race: arrayListOfRaces){
619              if (race.getRaceId()==(raceId)){
620                 for (Stage stage: race.getStages()){
621                    for (String key : setOfKeys){
622                       LocalTime adjustedElapsedTime =
                               stage.getRiderAdjustedElapsedTimeInStage(Integer.valueOf((key.split(" "))[1]));
623                       LocalTime currentDictionaryValue = riderTotalAdjustedElapsedTimeDictionary.get(key);
624                       Duration currentDictionaryValueDuration =
                               Duration.ofNanos(currentDictionaryValue.toNanoOfDay());
625                       long currentDictionaryValueLong = currentDictionaryValueDuration.toNanos();
```

```
                    LocalTime updatedDictionaryValue =
                        adjustedElapsedTime.plusNanos(currentDictionaryValueLong);
                    riderTotalAdjustedElapsedTimeDictionary.put(key, updatedDictionaryValue);
                }
            }ArrayList<LocalTime> arrayListOfTotalAdjustedElapsedTimesForEachRider = new ArrayList<>();
            for (String key : setOfKeys){
               arrayListOfTotalAdjustedElapsedTimesForEachRider.add(riderTotalAdjustedElapsedTimeDictionary.get(key));
            }Collections.sort(arrayListOfTotalAdjustedElapsedTimesForEachRider);
            //int[] array = arrayListOfTotalAdjustedElapsedTimesForEachRider.toArray(new
                int[arrayListOfTotalAdjustedElapsedTimesForEachRider.size()]);
            //return array;
            LocalTime[] arrayNew = new LocalTime[arrayListOfTotalAdjustedElapsedTimesForEachRider.size()];
            for (int i=0; i < arrayNew.length; i++) {
               arrayNew[i] = arrayListOfTotalAdjustedElapsedTimesForEachRider.get(i);
            }
            return arrayNew;
         }
      }
     }return new LocalTime[0];
  }

  @Override
  public int[] getRidersPointsInRace(int raceId) throws IDNotRecognisedException {
     ArrayList<Integer> arrayListOfRaceIDs = new ArrayList<>();
     for (Race race: arrayListOfRaces){
        arrayListOfRaceIDs.add(race.getRaceId());
     }
     if (!arrayListOfRaceIDs.contains(raceId)){
        throw new IDNotRecognisedException("The ID entered does not match to any race in the system");
     }
     ///
     ///
     ///
     ///
     ///
     //STUFF TO GET ARRAY OF ORDERED TOTAL ELAPSED TIMES
     int numberOfRiders = 0;
       Hashtable<String, LocalTime> riderTotalElapsedTimeDictionary = new Hashtable<String, LocalTime>();
     for (Race race: arrayListOfRaces){
          if (race.getRaceId()==(raceId)){
              if (race.getNumberOfStages()!=(0)){
                  numberOfRiders = (race.getStages().get(0)).getNumberOfResults();
                  for (Result result:((race.getStages().get(0)).getArrayListOfResults())){
                      if (!riderTotalElapsedTimeDictionary.containsKey("Rider
                          "+String.valueOf(result.getRiderId()))){
                          riderTotalElapsedTimeDictionary.put("Rider
                              "+String.valueOf(result.getRiderId()),LocalTime.of(0,0,0,0));
                      }
                  }
              }
          }
       }
       if (numberOfRiders==0){
          return new int[0];
       }
```

```
677         ArrayList<LocalTime> arrayListOfTotalElapsedTimesForEachRider = new ArrayList<>();
678           Set<String> setOfKeys = riderTotalElapsedTimeDictionary.keySet();
679           if (numberOfRiders==riderTotalElapsedTimeDictionary.size()){
680            for (Race race: arrayListOfRaces){
681               if (race.getRaceId()==(raceId)){
682                  for (Stage stage: race.getStages()){
683                     for (String key : setOfKeys){
684                        LocalTime elapsedTime = stage.getRiderElapsedTimeInStage(Integer.valueOf((key.split("
                              "))[1]));
685                        LocalTime currentDictionaryValue = riderTotalElapsedTimeDictionary.get(key);
686                        Duration currentDictionaryValueDuration =
                              Duration.ofNanos(currentDictionaryValue.toNanoOfDay());
687                        long currentDictionaryValueLong = currentDictionaryValueDuration.toNanos();
688                        LocalTime updatedDictionaryValue = elapsedTime.plusNanos(currentDictionaryValueLong);
689                        riderTotalElapsedTimeDictionary.put(key, updatedDictionaryValue);
690                     }
691                  }
692                  for (String key : setOfKeys){
693                     arrayListOfTotalElapsedTimesForEachRider.add(riderTotalElapsedTimeDictionary.get(key));
694                  }Collections.sort(arrayListOfTotalElapsedTimesForEachRider);
695               }
696            }
697         }
698         ///
699         //STUFF CONCERNING POINTS
700         ///
701         ///
702         ///
703         ///
704         ArrayList<Rider> allRiders = new ArrayList<>();
705         for (Team team:arrayListOfTeams){
706            for (Rider rider: team.arrayListOfRiders){
707               allRiders.add(rider);
708            }
709         }
710         int numberOfRiders2 = 0;
711           Hashtable<String, Integer> riderTotalPointsDictionary = new Hashtable<String, Integer>();
712         for (Race race: arrayListOfRaces){
713               if (race.getRaceId()==(raceId)){
714                   if (race.getNumberOfStages()!=(0)){
715                       numberOfRiders2 = (race.getStages().get(0)).getNumberOfResults();
716                       for (Result result:((race.getStages().get(0)).getArrayListOfResults())){
717                           if (!riderTotalPointsDictionary.containsKey("Rider
                                 "+String.valueOf(result.getRiderId()))){
718                              riderTotalPointsDictionary.put("Rider "+String.valueOf(result.getRiderId()),0);
719                           }
720                       }
721                   }
722               }
723           }
724           if (numberOfRiders2==0){
725               return new int[0];
726           }
727           Set<String> setOfKeys2 = riderTotalPointsDictionary.keySet();
728           if (numberOfRiders2==riderTotalPointsDictionary.size()){
```

```
729         for (Race race: arrayListOfRaces){
730            if (race.getRaceId()==(raceId)){
731               for (Stage stage: race.getStages()){
732                  for (String key : setOfKeys2){
733                     ArrayList<Integer> riderPointsInStage = stage.getRidersPointsInStage(allRiders);
734                     ArrayList<Integer> arrayListOfRiderIDsCorrespondingToRiderPointsInStageAL =
                           stage.getRidersRankInStage(allRiders);
735                     int indexOfID =
                           arrayListOfRiderIDsCorrespondingToRiderPointsInStageAL.indexOf(Integer.valueOf((key.split("
                           "))[1]));
736                     int pointsForGivenRiderInGivenStage = riderPointsInStage.get(indexOfID);
737                     int currentDictionaryValue = riderTotalPointsDictionary.get(key);
738                        int updatedDictionaryValue = currentDictionaryValue +=
                              pointsForGivenRiderInGivenStage;
739                     riderTotalPointsDictionary.put(key, updatedDictionaryValue);
740                  }
741               }
742            }
743         }
744      }ArrayList<Integer> riderTotalPointsSortedByTotalElapsedTime = new ArrayList<>();
745      ArrayList<Integer> riderIDsCorrespondingToTotalElapsedTimes = new ArrayList<>();
746      for (LocalTime elapsedTime: arrayListOfTotalElapsedTimesForEachRider){
747         for (String key: setOfKeys){
748            if (riderTotalElapsedTimeDictionary.get(key).equals(elapsedTime)){
749               riderIDsCorrespondingToTotalElapsedTimes.add(Integer.valueOf((key.split(" "))[1]));
750            }
751         }for (Integer riderId: riderIDsCorrespondingToTotalElapsedTimes){
752            riderTotalPointsSortedByTotalElapsedTime.add(riderTotalPointsDictionary.get("Rider "+
                  String.valueOf(riderId)));
753         }
754         int[] arrayNew = new int[riderTotalPointsSortedByTotalElapsedTime.size()];
755         for (int i=0; i < arrayNew.length; i++) {
756            arrayNew[i] = riderTotalPointsSortedByTotalElapsedTime.get(i);
757         }
758         return arrayNew;
759      }return new int[0];
760   }
761
762   @Override
763   public int[] getRidersMountainPointsInRace(int raceId) throws IDNotRecognisedException {
764      ArrayList<Integer> arrayListOfRaceIDs = new ArrayList<>();
765      for (Race race: arrayListOfRaces){
766         arrayListOfRaceIDs.add(race.getRaceId());
767      }
768      if (!arrayListOfRaceIDs.contains(raceId)){
769         throw new IDNotRecognisedException("The ID entered does not match to any race in the system");
770      }
771      ///
772      ///
773      ///
774      ///
775      ///
776      //STUFF TO GET ARRAY OF ORDERED TOTAL ELAPSED TIMES
777      int numberOfRiders = 0;
778        Hashtable<String, LocalTime> riderTotalElapsedTimeDictionary = new Hashtable<String, LocalTime>();
```

```
779         for (Race race: arrayListOfRaces){
780               if (race.getRaceId()==(raceId)){
781                   if (race.getNumberOfStages()!=(0)){
782                       numberOfRiders = (race.getStages().get(0)).getNumberOfResults();
783                       for (Result result:((race.getStages().get(0)).getArrayListOfResults())){
784                           if (!riderTotalElapsedTimeDictionary.containsKey("Rider
                                "+String.valueOf(result.getRiderId()))){
785                               riderTotalElapsedTimeDictionary.put("Rider
                                    "+String.valueOf(result.getRiderId()),LocalTime.of(0,0,0,0));
786                           }
787                       }
788                   }
789               }
790           }
791           if (numberOfRiders==0){
792               return new int[0];
793           }
794         ArrayList<LocalTime> arrayListOfTotalElapsedTimesForEachRider = new ArrayList<>();
795           Set<String> setOfKeys = riderTotalElapsedTimeDictionary.keySet();
796           if (numberOfRiders==riderTotalElapsedTimeDictionary.size()){
797            for (Race race: arrayListOfRaces){
798               if (race.getRaceId()==(raceId)){
799                  for (Stage stage: race.getStages()){
800                     for (String key : setOfKeys){
801                        LocalTime elapsedTime = stage.getRiderElapsedTimeInStage(Integer.valueOf((key.split("
                               "))[1]));
802                        LocalTime currentDictionaryValue = riderTotalElapsedTimeDictionary.get(key);
803                        Duration currentDictionaryValueDuration =
                                Duration.ofNanos(currentDictionaryValue.toNanoOfDay());
804                        long currentDictionaryValueLong = currentDictionaryValueDuration.toNanos();
805                        LocalTime updatedDictionaryValue = elapsedTime.plusNanos(currentDictionaryValueLong);
806                        riderTotalElapsedTimeDictionary.put(key, updatedDictionaryValue);
807                     }
808                  }
809                  for (String key : setOfKeys){
810                     arrayListOfTotalElapsedTimesForEachRider.add(riderTotalElapsedTimeDictionary.get(key));
811                  }Collections.sort(arrayListOfTotalElapsedTimesForEachRider);
812               }
813            }
814         }
815         ///
816         //STUFF CONCERNING POINTS
817         ///
818         ///
819         ///
820         ///
821         ArrayList<Rider> allRiders = new ArrayList<>();
822         for (Team team:arrayListOfTeams){
823            for (Rider rider: team.arrayListOfRiders){
824               allRiders.add(rider);
825            }
826         }
827         int numberOfRiders2 = 0;
828           Hashtable<String, Integer> riderTotalMountainPointsDictionary = new Hashtable<String, Integer>();
829         for (Race race: arrayListOfRaces){
```

```
            if (race.getRaceId()==(raceId)){
                if (race.getNumberOfStages()!=(0)){
                    numberOfRiders2 = (race.getStages().get(0)).getNumberOfResults();
                    for (Result result:((race.getStages().get(0)).getArrayListOfResults())){
                        if (!riderTotalMountainPointsDictionary.containsKey("Rider
                            "+String.valueOf(result.getRiderId()))){
                            riderTotalMountainPointsDictionary.put("Rider
                                "+String.valueOf(result.getRiderId()),0);
                        }
                    }
                }
            }
        }
        if (numberOfRiders2==0){
            return new int[0];
        }
        Set<String> setOfKeys2 = riderTotalMountainPointsDictionary.keySet();
        if (numberOfRiders2==riderTotalMountainPointsDictionary.size()){
         for (Race race: arrayListOfRaces){
            if (race.getRaceId()==(raceId)){
               for (Stage stage: race.getStages()){
                  for (String key : setOfKeys2){
                     ArrayList<Integer> riderMountainPointsInStage =
                         stage.getRidersMountainPointsInStage(allRiders);
                     ArrayList<Integer> arrayListOfRiderIDsCorrespondingToRiderMountainPointsInStageAL =
                         stage.getRidersRankInStage(allRiders);
                     int indexOfID =
                         arrayListOfRiderIDsCorrespondingToRiderMountainPointsInStageAL.indexOf(Integer.valueOf((key.spl
                         "))[1]));
                     int mountainPointsForGivenRiderInGivenStage = riderMountainPointsInStage.get(indexOfID);
                     int currentDictionaryValue = riderTotalMountainPointsDictionary.get(key);
                        int updatedDictionaryValue = currentDictionaryValue +=
                            mountainPointsForGivenRiderInGivenStage;
                     riderTotalMountainPointsDictionary.put(key, updatedDictionaryValue);
                  }
               }
            }
         }
      }ArrayList<Integer> riderTotalMountainPointsSortedByTotalElapsedTime = new ArrayList<>();
      ArrayList<Integer> riderIDsCorrespondingToTotalElapsedTimes = new ArrayList<>();
      for (LocalTime elapsedTime: arrayListOfTotalElapsedTimesForEachRider){
         for (String key: setOfKeys){
            if (riderTotalElapsedTimeDictionary.get(key).equals(elapsedTime)){
               riderIDsCorrespondingToTotalElapsedTimes.add(Integer.valueOf((key.split(" "))[1]));
            }
         }for (Integer riderId: riderIDsCorrespondingToTotalElapsedTimes){
            riderTotalMountainPointsSortedByTotalElapsedTime.add(riderTotalMountainPointsDictionary.get("Rider
                "+ String.valueOf(riderId)));
         }
         int[] arrayNew = new int[riderTotalMountainPointsSortedByTotalElapsedTime.size()];
         for (int i=0; i < arrayNew.length; i++) {
            arrayNew[i] = riderTotalMountainPointsSortedByTotalElapsedTime.get(i);
         }
         return arrayNew;
      }return new int[0];
```

```
877     }
878
879     @Override
880     public int[] getRidersGeneralClassificationRank(int raceId) throws IDNotRecognisedException {
881        ArrayList<Integer> arrayListOfRaceIDs = new ArrayList<>();
882        for (Race race: arrayListOfRaces){
883           arrayListOfRaceIDs.add(race.getRaceId());
884        }
885        if (!arrayListOfRaceIDs.contains(raceId)){
886           throw new IDNotRecognisedException("The ID entered does not match to any race in the system");
887        }
888          int numberOfRiders = 0;
889          Hashtable<String, LocalTime> riderTotalAdjustedElapsedTimeDictionary = new Hashtable<String,
                LocalTime>();
890        for (Race race: arrayListOfRaces){
891              if (race.getRaceId()==(raceId)){
892                  if (race.getNumberOfStages()!=(0)){
893                      numberOfRiders = (race.getStages().get(0)).getNumberOfResults();
894                      for (Result result:((race.getStages().get(0)).getArrayListOfResults())){
895                          if (!riderTotalAdjustedElapsedTimeDictionary.containsKey("Rider
                                "+String.valueOf(result.getRiderId()))){
896                             riderTotalAdjustedElapsedTimeDictionary.put("Rider
                                  "+String.valueOf(result.getRiderId()),LocalTime.of(0,0,0,0));
897                          }
898                      }
899                  }
900              }
901          }
902          if (numberOfRiders==0){
903              return new int[0];
904          }
905          Set<String> setOfKeys = riderTotalAdjustedElapsedTimeDictionary.keySet();
906          if (numberOfRiders==riderTotalAdjustedElapsedTimeDictionary.size()){
907           for (Race race: arrayListOfRaces){
908              if (race.getRaceId()==(raceId)){
909                 for (Stage stage: race.getStages()){
910                    for (String key : setOfKeys){
911                       LocalTime adjustedElapsedTime =
                             stage.getRiderAdjustedElapsedTimeInStage(Integer.valueOf((key.split(" "))[1]));
912                       LocalTime currentDictionaryValue = riderTotalAdjustedElapsedTimeDictionary.get(key);
913                       //long currentDictionaryValueDuration =
                             Duration.ofNanos(currentDictionaryValue.toNanoOfDay());
914                       Duration currentDictionaryValueDuration =
                             Duration.ofNanos(currentDictionaryValue.toNanoOfDay());
915                       long currentDictionaryValueLong = currentDictionaryValueDuration.toNanos();
916                       LocalTime updatedDictionaryValue =
                             adjustedElapsedTime.plusNanos(currentDictionaryValueLong);
917                       //LocalTime updatedDictionaryValue =
                             adjustedElapsedTime.plusNanos(currentDictionaryValueDuration);
918                       riderTotalAdjustedElapsedTimeDictionary.put(key, updatedDictionaryValue);
919                    }
920                 }ArrayList<LocalTime> arrayListOfTotalAdjustedElapsedTimesForEachRider = new ArrayList<>();
921                 for (String key : setOfKeys){
922                    arrayListOfTotalAdjustedElapsedTimesForEachRider.add(riderTotalAdjustedElapsedTimeDictionary.get(key))
923                 }Collections.sort(arrayListOfTotalAdjustedElapsedTimesForEachRider);
```

```
            ArrayList<Integer> arrayListOfIDsSortedByTAE = new ArrayList<>();
            for (LocalTime j: arrayListOfTotalAdjustedElapsedTimesForEachRider){
               for (String key : setOfKeys){
                  if (!arrayListOfIDsSortedByTAE.contains(Integer.valueOf((key.split(" "))[1]))){
                     if (riderTotalAdjustedElapsedTimeDictionary.get(key).equals(j)){
                        arrayListOfIDsSortedByTAE.add(Integer.valueOf((key.split(" "))[1]));
                        break;
                     }
                  }
               }
            }//int[] array = arrayListOfIDsSortedByTAE.toArray(new int[arrayListOfIDsSortedByTAE.size()]);
            //return array;
            int[] arrayNew = new int[arrayListOfIDsSortedByTAE.size()];
            for (int i=0; i < arrayNew.length; i++) {
               arrayNew[i] = arrayListOfIDsSortedByTAE.get(i);
            }
            return arrayNew;
         }
      }
     }return new int[0];
  }

  @Override
  public int[] getRidersPointClassificationRank(int raceId) throws IDNotRecognisedException {
     ArrayList<Integer> arrayListOfRaceIDs = new ArrayList<>();
     for (Race race: arrayListOfRaces){
        arrayListOfRaceIDs.add(race.getRaceId());
     }
     if (!arrayListOfRaceIDs.contains(raceId)){
        throw new IDNotRecognisedException("The ID entered does not match to any race in the system");
     }
     ArrayList<Rider> allRiders = new ArrayList<>();
     for (Team team:arrayListOfTeams){
        for (Rider rider: team.arrayListOfRiders){
           allRiders.add(rider);
        }
     }
     int numberOfRiders = 0;
       Hashtable<String, Integer> riderTotalPointsDictionary = new Hashtable<String, Integer>();
     for (Race race: arrayListOfRaces){
           if (race.getRaceId()==(raceId)){
               if (race.getNumberOfStages()!=(0)){
                   numberOfRiders = (race.getStages().get(0)).getNumberOfResults();
                   for (Result result:((race.getStages().get(0)).getArrayListOfResults())){
                       if (!riderTotalPointsDictionary.containsKey("Rider
                            "+String.valueOf(result.getRiderId()))){
                           riderTotalPointsDictionary.put("Rider "+String.valueOf(result.getRiderId()),0);
                       }
                   }
               }
           }
       }
       if (numberOfRiders==0){
           return new int[0];
       }
```

```
978         Set<String> setOfKeys = riderTotalPointsDictionary.keySet();
979         if (numberOfRiders==riderTotalPointsDictionary.size()){
980          for (Race race: arrayListOfRaces){
981             if (race.getRaceId()==(raceId)){
982                for (Stage stage: race.getStages()){
983                   for (String key : setOfKeys){
984                      ArrayList<Integer> riderPointsInStage = stage.getRidersPointsInStage(allRiders);
985                      ArrayList<Integer> arrayListOfRiderIDsCorrespondingToRiderPointsInStageAL =
                             stage.getRidersRankInStage(allRiders);
986                      int indexOfID =
                             arrayListOfRiderIDsCorrespondingToRiderPointsInStageAL.indexOf(Integer.valueOf((key.split("
                             "))[1]));
987                      int pointsForGivenRiderInGivenStage = riderPointsInStage.get(indexOfID);
988                      int currentDictionaryValue = riderTotalPointsDictionary.get(key);
989                         int updatedDictionaryValue = currentDictionaryValue +=
                                pointsForGivenRiderInGivenStage;
990                      riderTotalPointsDictionary.put(key, updatedDictionaryValue);
991                   }
992                }ArrayList<Integer> arrayListOfTotalPointsForEachRider = new ArrayList<>();
993                for (String key : setOfKeys){
994                   arrayListOfTotalPointsForEachRider.add(riderTotalPointsDictionary.get(key));
995                }Collections.sort(arrayListOfTotalPointsForEachRider, Collections.reverseOrder());
996                ArrayList<Integer> arrayListOfIDsSortedByPoints = new ArrayList<>();
997                for (int j: arrayListOfTotalPointsForEachRider){
998                   for (String key : setOfKeys){
999                      if (!arrayListOfIDsSortedByPoints.contains(Integer.valueOf((key.split(" "))[1]))){
1000                        if (riderTotalPointsDictionary.get(key)==(j)){
1001                           arrayListOfIDsSortedByPoints.add(Integer.valueOf((key.split(" "))[1]));
1002                           break;
1003                        }
1004                     }
1005                  }
1006                  }
1007               int[] arrayNew = new int[arrayListOfIDsSortedByPoints.size()];
1008               for (int i=0; i < arrayNew.length; i++) {
1009                  arrayNew[i] = arrayListOfIDsSortedByPoints.get(i);
1010               }
1011               return arrayNew;
1012            }
1013         }
1014      }return new int[0];
1015   }
1016
1017   @Override
1018   public int[] getRidersMountainPointClassificationRank(int raceId) throws IDNotRecognisedException {
1019      ArrayList<Integer> arrayListOfRaceIDs = new ArrayList<>();
1020      for (Race race: arrayListOfRaces){
1021         arrayListOfRaceIDs.add(race.getRaceId());
1022      }
1023      if (!arrayListOfRaceIDs.contains(raceId)){
1024         throw new IDNotRecognisedException("The ID entered does not match to any race in the system");
1025      }
1026      ArrayList<Rider> allRiders = new ArrayList<>();
1027      for (Team team:arrayListOfTeams){
1028         for (Rider rider: team.arrayListOfRiders){
```

```
            allRiders.add(rider);
        }
    }
    int numberOfRiders = 0;
      Hashtable<String, Integer> riderTotalMountainPointsDictionary = new Hashtable<String, Integer>();
    for (Race race: arrayListOfRaces){
         if (race.getRaceId()==(raceId)){
             if (race.getNumberOfStages()!=(0)){
                 numberOfRiders = (race.getStages().get(0)).getNumberOfResults();
                 for (Result result:((race.getStages().get(0)).getArrayListOfResults())){
                     if (!riderTotalMountainPointsDictionary.containsKey("Rider
                         "+String.valueOf(result.getRiderId()))){
                         riderTotalMountainPointsDictionary.put("Rider
                             "+String.valueOf(result.getRiderId()),0);
                     }
                 }
             }
         }
      }
      if (numberOfRiders==0){
         return new int[0];
      }
      Set<String> setOfKeys = riderTotalMountainPointsDictionary.keySet();
      if (numberOfRiders==riderTotalMountainPointsDictionary.size()){
       for (Race race: arrayListOfRaces){
         if (race.getRaceId()==(raceId)){
            for (Stage stage: race.getStages()){
               for (String key : setOfKeys){
                  ArrayList<Integer> riderMountainPointsInStage =
                      stage.getRidersMountainPointsInStage(allRiders);
                  ArrayList<Integer> arrayListOfRiderIDsCorrespondingToRiderMountainPointsInStageAL =
                      stage.getRidersRankInStage(allRiders);
                  int indexOfID =
                      arrayListOfRiderIDsCorrespondingToRiderMountainPointsInStageAL.indexOf(Integer.valueOf((key.spl
                      "))[1]));
                  int mountainPointsForGivenRiderInGivenStage = riderMountainPointsInStage.get(indexOfID);
                  int currentDictionaryValue = riderTotalMountainPointsDictionary.get(key);
                     int updatedDictionaryValue = currentDictionaryValue +=
                         mountainPointsForGivenRiderInGivenStage;
                  riderTotalMountainPointsDictionary.put(key, updatedDictionaryValue);
               }
            }ArrayList<Integer> arrayListOfTotalMountainPointsForEachRider = new ArrayList<>();
            for (String key : setOfKeys){
               arrayListOfTotalMountainPointsForEachRider.add(riderTotalMountainPointsDictionary.get(key));
            }Collections.sort(arrayListOfTotalMountainPointsForEachRider, Collections.reverseOrder());
            ArrayList<Integer> arrayListOfIDsSortedByMountainPoints = new ArrayList<>();
            for (int j: arrayListOfTotalMountainPointsForEachRider){
               for (String key : setOfKeys){
                  if (!arrayListOfIDsSortedByMountainPoints.contains(Integer.valueOf((key.split("
                      "))[1]))){
                     if (riderTotalMountainPointsDictionary.get(key)==(j)){
                        arrayListOfIDsSortedByMountainPoints.add(Integer.valueOf((key.split(" "))[1]));
                        break;
                     }
                  }
```

```
                }
                }
            int[] arrayNew = new int[arrayListOfIDsSortedByMountainPoints.size()];
            for (int i=0; i < arrayNew.length; i++) {
                arrayNew[i] = arrayListOfIDsSortedByMountainPoints.get(i);
            }
            return arrayNew;
        }
      }
    }return new int[0];
  }
}
```

2 Race.java

```
package cycling;

import java.io.Serializable;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Collections;

/**
 * The Race class stores all the information related to a race, and stores an arraylist of the stages for
     the race.
 */
public class Race implements Serializable {
    public static int raceCounter = 0;
    private int raceId;
    private String name;
    private String description;
    private int numberOfStages;
    private double totalLength;
    ArrayList<Stage> arrayListOfStages = new ArrayList<>();

    /**
     * Constructor for the Race class.
     * @param name The name of the race.
     * @param description The description of the race.
     */
    public Race(String name, String description) {
        this.name = name;
        this.description = description;
        raceCounter += 1;
        raceId = raceCounter;
    }

    /**
     * Gets the ID for the race.
     * @return The race ID.
     */
    public int getRaceId() {
        return raceId;
    }
```

```
39
40      /**
41       * Gets the name of the race.
42       * @return The race name.
43       */
44      public String getName() {
45          return name;
46      }
47
48      /**
49       * Gets the description of the race.
50       * @return The description of the race.
51       */
52      public String getDescription() {
53          return description;
54      }
55
56      /**
57       * Gets the number of stages that are in the race.
58       * @return The number of stages in the race.
59       */
60      public int getNumberOfStages() {
61          numberOfStages = arrayListOfStages.size();
62          return numberOfStages;
63      }
64
65      /**
66       * Gets the total length for a race in kilometres.
67       * @return The total length of the race.
68       */
69      public double getTotalLength() {
70          for (Stage stage:arrayListOfStages) {
71              totalLength += stage.getStageLength();
72          }return totalLength;
73      }
74
75      /**
76       * Gets all the stages in the race.
77       * @return A list of the stages that belong to the race.
78       */
79      public ArrayList<Stage> getStages() {
80          return arrayListOfStages;
81      }
82
83      /**
84       * Creates a new stage and adds it to the race.
85       * @param raceId The ID of the race which the stage is added to.
86       * @param stageName The name for the stage.
87       * @param description A description for the stage.
88       * @param length The length of the stage in kilometres.
89       * @param startTime The date and time in which the stage will be raced.
90       * @param type The type of the stage.
91       */
92      public int createStage(int raceId, String stageName, String description, double length,
93                             LocalDateTime startTime, StageType type) {
```

```
94          Stage newStage = new Stage(raceId, stageName, description, length, startTime, type);
95          arrayListOfStages.add(newStage);
96          return newStage.getStageId();
97      }
98
99      /**
100      * Removes a stage from a race and all its related data.
101      * @param stageId The ID of the stage being removed.
102      */
103     public void removeStage(int stageId) {
104         for (Stage i:arrayListOfStages){
105             if ((i.getStageId()) == stageId){
106                 arrayListOfStages.remove(i);
107                 break;
108             }
109         }
110     }
111
112     /**
113      * Orders the stages based on which stage starts soonest to latest.
114      * @return A list of stage ID's ordered by when they start.
115      */
116     public ArrayList<Integer> getRaceStages(){
117         ArrayList<LocalDateTime> arrayListOfStageStartDateTimes = new ArrayList<>();
118         for (Stage stage:arrayListOfStages) {
119             arrayListOfStageStartDateTimes.add(stage.getStartTime());
120         }
121         Collections.sort(arrayListOfStageStartDateTimes);
122         ArrayList<Integer> arrayListOfSortedStageIDs = new ArrayList<>();
123         for (LocalDateTime i:arrayListOfStageStartDateTimes){
124             for (Stage k:arrayListOfStages){
125                 if (i.equals(k.getStartTime())){
126                     if (!arrayListOfSortedStageIDs.contains(k.getStageId())){
127                         arrayListOfSortedStageIDs.add(k.getStageId());
128                     }
129                 }
130             }
131         }
132         return arrayListOfSortedStageIDs;
133     }
134 }
```

3 Stage.java

```
1  package cycling;
2
3  import java.io.Serializable;
4  import java.security.Identity;
5  import java.time.LocalTime;
6  import java.util.ArrayDeque;
7  import java.util.ArrayList;
8  import java.time.LocalDateTime;
9  import java.util.Collections;
10 import java.util.Hashtable;
```

```java
import java.lang.reflect.Array;

/**
 * The Stage class stores all the information related to a stage, and stores arraylists of results and segments for the given stage.
 */
public class Stage implements Serializable {
    public static int stageCounter = 0;
    private int stageId;//unique stage identifier
    private int raceId;
    private String stageName;//name of stage
    private String description;//description of stage
    private double length;
    private LocalDateTime startTime;
    private StageType type;
    private String state = "Not waiting on results";
    ArrayList<Result> arrayListOfResults = new ArrayList<>();//arraylist of all rider objects for given race
    ArrayList<Segment> arrayListOfSegments = new ArrayList<>();


    /**
     * Constructor for the Stage class.
     * @param raceId The ID of the race this stage object belongs to.
     * @param description The description of the stage.
     * @param length The length of the stage in kilometres.
     * @param startTime The date and time in which the stage will be raced.
     * @param type The type of the stage. This is used to determine the amount of points given to the winner.
     */
    public Stage(int raceId, String stageName, String description, double length, LocalDateTime startTime, StageType type) {
        this.raceId = raceId;
        this.stageName = stageName;
        this.description = description;
        this.length = length;
        this.startTime = startTime;
        this.type = type;
        stageCounter+=1;
        stageId = stageCounter;
    }

    /**
     * Creates and returns a Hashtable dictionary with keys representing each rider in the stage and
     * their respective values representing their Sprinter's Classification points in the stage.
     * Values for all keys are initialised as 0 here.
     * @return The default starter Hashtable of rider keys each with a value of 0.
     */
    public Hashtable<String, Integer> createRiderSprintersPointsDictionary(){
        Hashtable<String, Integer> riderSprintersPointsDictionary = new Hashtable<String, Integer>();
        for (Result result:arrayListOfResults){
            if (!riderSprintersPointsDictionary.containsKey("Rider "+String.valueOf(result.getRiderId()))){
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(result.getRiderId()),0);
            }
        }
        return riderSprintersPointsDictionary;
    }
```

```
64
65      /**
66       * Creates and returns a Hashtable dictionary with keys representing each rider in the stage and
67       * their respective values representing their KOM Classification points in the stage.
68       * Values for all keys are initialised as 0 here.
69       * @return The default starter Hashtable of rider keys each with a value of 0.
70       */
71      public Hashtable<String, Integer> createRiderKOMPointsDictionary(){
72          Hashtable<String, Integer> riderKOMPointsDictionary = new Hashtable<String, Integer>();
73          for (Result result:arrayListOfResults){
74              if (!riderKOMPointsDictionary.containsKey("Rider "+String.valueOf(result.getRiderId()))){
75                  riderKOMPointsDictionary.put("Rider "+String.valueOf(result.getRiderId()),0);
76              }
77          }
78          return riderKOMPointsDictionary;
79      }
80
81      /**
82       * Gets the length for the stage.
83       * @return The stage length in kilometres.
84       */
85      public double getStageLength() {
86          return length;
87      }
88
89      /**
90       * Gets the ID for the stage.
91       * @return The stage ID.
92       */
93      public int getStageId(){
94          return stageId;
95      }
96
97      /**
98       * Gets the ID for the race the stage object belongs to.
99       * @return The race ID the stage object belongs to.
100      */
101     public int getRaceId(){
102         return raceId;
103     }
104
105     /**
106      * Gets the stage name.
107      * @return The stage name.
108      */
109     public String getStageName(){
110         return stageName;
111     }
112
113     /**
114      * Gets the start date and time in which the stage will be raced.
115      * @return The start date and time in which the stage will be raced..
116      */
117     public LocalDateTime getStartTime(){
118         return startTime;
```

```
119     }
120
121     /**
122      * Gets the stage description.
123      * @return The stage description.
124      */
125     public String getDescription(){
126         return description;
127     }
128
129     /**
130      * Gets the number of result objects (held in the arraylist of results) within the stage.
131      * @return The number of result objects (held in the arraylist of results) within the stage.
132      */
133     public int getNumberOfResults(){
134         return arrayListOfResults.size();
135     }
136
137     /**
138      * Gets the number of Result objects (held in the arraylist of results) within the stage.
139      * @return The number of result objects (held in the arraylist of results) within the stage.
140      */
141     public ArrayList<Result> getArrayListOfResults(){
142         return arrayListOfResults;
143     }
144
145     /**
146      * Retrieves the list of segment (mountains and sprints) IDs for the Segment objects of the stage
147      * @return The list of segment IDs ordered (from first to last) by their location in the stage.
148      */
149     public ArrayList<Integer> getStageSegments(){
150         ArrayList<Double> arrayListOfSegmentLocations = new ArrayList<>();
151         for (Segment segment:arrayListOfSegments){
152             arrayListOfSegmentLocations.add(segment.getLocation());
153         }
154         Collections.sort(arrayListOfSegmentLocations);
155         ArrayList<Integer> arrayListOfSegmentIDsSortedByLocation = new ArrayList<>();
156         ArrayList<Double> oldArrayListOfSegmentLocations = new ArrayList<>();
157         ArrayList<Integer> tempArrayOfIndexes = new ArrayList<>();
158         for (Segment segment:arrayListOfSegments){
159             oldArrayListOfSegmentLocations.add(segment.getLocation());
160         }
161         for (Double segmentLocation:arrayListOfSegmentLocations){
162             int locationIndex = oldArrayListOfSegmentLocations.indexOf(segmentLocation);
163             tempArrayOfIndexes.add(locationIndex);
164         }
165         for (int index:tempArrayOfIndexes){
166             Segment exampleSegment = arrayListOfSegments.get(index);
167             int id = exampleSegment.getSegmentId();
168             arrayListOfSegmentIDsSortedByLocation.add(id);
169         }
170
171         return arrayListOfSegmentIDsSortedByLocation;
172     }
173
```

```
174     /**
175      * Concludes the preparation of a stage. After conclusion, the stage's state
176     * should be "waiting for results".
177      */
178     public void concludeStagePreparation(){
179         state = "waiting for results";
180 
181     }
182 
183     /**
184      * Gets the state of the stage.
185      * @return The state of the stage.
186      */
187     public String getStageState(){
188         return state;
189     }
190 
191     /**
192      * Gets the type of the stage.
193      * @return The type of the stage.
194      */
195     public StageType getStageType(){
196         return type;
197     }
198 
199     /**
200      * Adds a climb segment to a stage.
201      * @param stageId The ID of the stage to which the climb segment is being added.
202      * @param location The kilometre location where the climb finishes within the stage.
203      * @param type The category of the climb - {@link SegmentType#C4}, {@link SegmentType#C3},
204      *                                       {@link SegmentType#C2}, {@link SegmentType#C1},
205      *                                      or {@link SegmentType#HC}
206      * @param averageGradient The average gradient for the climb.
207      * @param length The length of the climb in kilometres.
208      * @return The ID of the Segment object created.
209      *
210      */
211     public int addCategorizedClimbToStage (int stageId, Double location, SegmentType type, Double
         averageGradient, Double length) {
212         Double locationnew = location;
213         SegmentType typenew = type;
214         Double averageGradientnew = averageGradient;
215         Double lengthnew = length;
216         CategorizedClimb newClimb = new CategorizedClimb(stageId, locationnew, typenew, averageGradientnew,
             lengthnew);
217         arrayListOfSegments.add(newClimb);
218         return newClimb.getSegmentId();
219     }
220 
221     /**
222      * Adds an intermediate sprint to a stage.
223      * @param stageId The ID of the stage to which the intermediate sprint segment is being added.
224      * @param location The kilometre location where the intermediate sprint finishes within the stage.
225      * @return The ID of the Segment object created.
226      */
```

```
227     public int addIntermediateSprintToStage (int stageId, double location) {
228         double locationnew = location;
229         IntermediateSprint newSprint = new IntermediateSprint(stageId, locationnew);
230         arrayListOfSegments.add(newSprint);
231         return newSprint.getSegmentId();
232     }
233 
234     /**
235      * Removes a segment from a stage.
236      * @param segmentid The ID of the Segment object to be removed from the arraylist of segments within the
            stage.
237      */
238     public void removeSegment(int segmentid){
239         for (Segment i: arrayListOfSegments){
240             if (i.getSegmentId()==(segmentid)){
241                 arrayListOfSegments.remove(i);
242                 break;
243             }
244         }
245     }
246 
247     /**
248      * Record the times of a rider in a stage.
249      * @param stageId The ID of the stage the result refers to.
250      * @param riderId The ID of the rider the result refers to.
251      * @param checkpoints An array of times at which the rider reached each of the
252     *                  segments of the stage, including the start time and the
253     *                  finish line.
254      */
255     public void registerRiderResultsInStage(int stageId, int riderId, LocalTime[] checkpoints) {
256         int stageIdnew = stageId;
257         int riderIdnew = riderId;
258         LocalTime[] checkpointsnew = checkpoints;
259         Result newResults = new Result(stageIdnew, riderIdnew, checkpointsnew);
260         arrayListOfResults.add(newResults);
261     }
262 
263     /**
264      * Get the times of a rider in a stage.
265      * @param riderId The ID of the rider a given result refers to.
266      * @return The array of times at which the rider reached each of the segments of
267     *         the stage and the total elapsed time. The elapsed time is the
268     *         difference between the finish time and the start time.
269      *
270      *         An empty array is returned if there are no results registered for the
271      *         given stage; i.e, the array list of result objects of the stage object
272      *         is empty.
273      */
274     public ArrayList<LocalTime> getRiderResultsInStage(int riderId){
275         for (Result i:arrayListOfResults){
276             if ((i.getRiderId())==(riderId)){
277                 LocalTime[] checkPointsArray = i.getCheckpoints();
278                 ArrayList<LocalTime> checkpointsArrayList = new ArrayList<>();
279                 for (LocalTime checkpoint:checkPointsArray){
280                     checkpointsArrayList.add(checkpoint);
```

```
281                 }
282                 checkpointsArrayList.add(i.getElapsedTime());
283                 return checkpointsArrayList;
284             }
285         }return new ArrayList<LocalTime>();
286     }
287 
288     /**
289      * Gets the adjusted elapsed time for a rider in the stage.
290     * @param riderId The ID of the rider a given result refers to.
291     * @return The adjusted elapsed time for the rider in the stage.
292      *
293      *         Null if there is no result registered for
294      *         the rider in the stage.
295      */
296     public LocalTime getRiderAdjustedElapsedTimeInStage(int riderId) {
297         for (Result i:arrayListOfResults){
298             if ((i.getRiderId())==(riderId)){
299                 return i.getAdjustedElapsedTime(arrayListOfResults);
300             }
301         }return null;
302     }
303 
304     /**
305      * Gets the elapsed time for a rider in the stage.
306     * @param riderId The ID of the rider a given result refers to.
307     * @return The elapsed time for the rider in the stage.
308      *
309      *         Null if there is no result registered for
310      *         the rider in the stage.
311      */
312     public LocalTime getRiderElapsedTimeInStage(int riderId){
313         for (Result i:arrayListOfResults){
314             if ((i.getRiderId())==(riderId)){
315                 return i.getElapsedTime();
316             }
317         }return null;
318     }
319 
320 
321     /**
322      * Removes the stage results from the rider.
323     * @param riderId The ID of the rider a given result refers to.
324      */
325     public void deleteRiderResultsInStage(int riderId) {
326         for (Result i:arrayListOfResults){
327             if ((i.getRiderId())==(riderId)){
328                 arrayListOfResults.remove(i);
329                 break;
330             }
331         }
332     }
333 
334     /**
335      * Get the riders finished position in a a stage.
```

```
     * @param allRiders An arraylist of all riders created in the cycling portal for the current race/stage
    * @return A list of riders' ID sorted by their elapsed time.
     *
     *        An empty list if there is no result for the stage.
     */
    public ArrayList<Integer> getRidersRankInStage (ArrayList<Rider> allRiders) {
        ArrayList<LocalTime> arrayListOfRiderElapsedTimes = new ArrayList<>();
        if (arrayListOfResults.isEmpty()){
            return (new ArrayList<Integer>());
        }
        for(Result j:arrayListOfResults){
            arrayListOfRiderElapsedTimes.add(j.getElapsedTime());
        }
        Collections.sort(arrayListOfRiderElapsedTimes);
        ArrayList<Integer> arrayListOfSortedRiderIDs = new ArrayList<>();
        for(LocalTime z:arrayListOfRiderElapsedTimes){
            for (Result k:arrayListOfResults){
                if (z.equals(k.getElapsedTime())){
                    arrayListOfSortedRiderIDs.add(k.getRiderId());
                }
            }
        }
        return arrayListOfSortedRiderIDs;
    }

    /**
     * Get the adjusted elapsed times of riders in a stage.
     * @param allRiders An arraylist of all riders created in the cycling portal for the current race/stage
    * @return The ranked list of adjusted elapsed times sorted by their finish time.
     *
     *        An empty list if there is no result for the stage.
     */
    public ArrayList<LocalTime> getRankedAdjustedElapsedTimesInStage(ArrayList<Rider> allRiders) {
        if (arrayListOfResults.isEmpty()){
            return (new ArrayList<LocalTime>());
        }
        ArrayList<LocalTime> arrayListOfRiderFinishTimes = new ArrayList<>();

        for(Result j:arrayListOfResults){
            arrayListOfRiderFinishTimes.add(j.getFinishTime());
        }
        Collections.sort(arrayListOfRiderFinishTimes);
        ArrayList<LocalTime> arrayListOfSortedAdjustedElapsedTimes = new ArrayList<>();
        for(LocalTime z:arrayListOfRiderFinishTimes){
            for (Result k:arrayListOfResults){
                if (z.equals(k.getFinishTime())){
                    arrayListOfSortedAdjustedElapsedTimes.add(k.getAdjustedElapsedTime(arrayListOfResults));
                }
            }
        }return arrayListOfSortedAdjustedElapsedTimes;
    }

    /**
     * Get the number of points obtained by each rider in a stage.
     * @param allRiders An arraylist of all riders created in the cycling portal for the current race/stage
```

```
391     * @return @return The ranked list of points each rider received in the stage, sorted
392     *         by their elapsed time.
393      *
394      *         An empty list if there is no result for the stage.
395      */
396     public ArrayList<Integer> getRidersPointsInStage(ArrayList<Rider> allRiders) {
397         if (arrayListOfResults.isEmpty()){
398             return (new ArrayList<Integer>());
399         }
400         ArrayList<LocalTime> arrayListOfRiderElapsedTimes = new ArrayList<>();
401         ArrayList<LocalTime> arrayListOfRiderFinishTimes = new ArrayList<>();
402 
403         for(Result j:arrayListOfResults){
404             arrayListOfRiderElapsedTimes.add(j.getElapsedTime());
405         }
406         for(Result j:arrayListOfResults){
407             arrayListOfRiderFinishTimes.add(j.getFinishTime());
408         }
409 
410         Collections.sort(arrayListOfRiderElapsedTimes);
411         //time trial handler
412         if (this.type.equals(StageType.TT)){
413             ArrayList<Integer> arrayListOfSortedRiderIDs = new ArrayList<>();
414             for(LocalTime z:arrayListOfRiderElapsedTimes){
415                 for (Result k:arrayListOfResults){
416                     if (z.equals(k.getElapsedTime())){
417                         arrayListOfSortedRiderIDs.add(k.getRiderId());
418                     }
419                 }
420             }
421             Hashtable <String, Integer> riderSprintersPointsDictionary =
                     this.createRiderSprintersPointsDictionary();
422             for (int id: arrayListOfSortedRiderIDs){
423                 for (Result result: arrayListOfResults){
424                     if(result.getRiderId()==(id)){
425                         int currentDictionaryValue = riderSprintersPointsDictionary.get("Rider
                                 "+String.valueOf(id));
426                         if
                                 ((arrayListOfSortedRiderIDs.indexOf(id)+1)>15||(arrayListOfSortedRiderIDs.indexOf(id)+1)<1){
427                             break;
428                         }else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==1){
429                             int updatedDictionaryValue = currentDictionaryValue+=20;
430                             riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                                     updatedDictionaryValue);
431                         }
432                         else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==2){
433                             int updatedDictionaryValue = currentDictionaryValue+=17;
434                             riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                                     updatedDictionaryValue);
435                         }
436                         else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==3){
437                             int updatedDictionaryValue = currentDictionaryValue+=15;
438                             riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                                     updatedDictionaryValue);
439                         }
```

```
440                 else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==4){
441                     int updatedDictionaryValue = currentDictionaryValue+=13;
442                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
443                 }
444                 else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==5){
445                     int updatedDictionaryValue = currentDictionaryValue+=11;
446                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
447                 }
448                 else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==6){
449                     int updatedDictionaryValue = currentDictionaryValue+=10;
450                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
451                 }
452                 else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==7){
453                     int updatedDictionaryValue = currentDictionaryValue+=9;
454                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
455                 }
456                 else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==8){
457                     int updatedDictionaryValue = currentDictionaryValue+=8;
458                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
459                 }
460                 else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==9){
461                     int updatedDictionaryValue = currentDictionaryValue+=7;
462                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
463                 }
464                 else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==10){
465                     int updatedDictionaryValue = currentDictionaryValue+=6;
466                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
467                 }
468                 else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==11){
469                     int updatedDictionaryValue = currentDictionaryValue+=5;
470                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
471                 }
472                 else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==12){
473                     int updatedDictionaryValue = currentDictionaryValue+=4;
474                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
475                 }
476                 else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==13){
477                     int updatedDictionaryValue = currentDictionaryValue+=3;
478                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
479                 }
480                 else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==14){
481                     int updatedDictionaryValue = currentDictionaryValue+=2;
482                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
483                 }
```

```
                    else if((arrayListOfSortedRiderIDs.indexOf(id)+1)==15){
                        int updatedDictionaryValue = currentDictionaryValue+=1;
                        riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
                    }
                }
            }
        }ArrayList<Integer> arrayListOfPointsCorrespondingToRiderIDsSortedByElapsedTime = new
            ArrayList<>();
        for (int id:arrayListOfSortedRiderIDs){
            arrayListOfPointsCorrespondingToRiderIDsSortedByElapsedTime.add(riderSprintersPointsDictionary.get("Rid
                "+String.valueOf(id)));
        }
        return arrayListOfPointsCorrespondingToRiderIDsSortedByElapsedTime;
    }
    //finish position handler
    Collections.sort(arrayListOfRiderFinishTimes);

    Hashtable <String, Integer> riderSprintersPointsDictionary =
        this.createRiderSprintersPointsDictionary();
    for (Result result:arrayListOfResults){
        int givenRiderId = result.getRiderId();
        LocalTime givenFinishTime = result.getFinishTime();
        int positionOfGivenRiderInStage = arrayListOfRiderFinishTimes.indexOf(givenFinishTime);
        int currentDictionaryValue = riderSprintersPointsDictionary.get("Rider
            "+String.valueOf(givenRiderId));
        if (this.type.equals(StageType.FLAT)){
            if (positionOfGivenRiderInStage+1>15||positionOfGivenRiderInStage+1<1){
                break;
            }else if(positionOfGivenRiderInStage+1==1){
                int updatedDictionaryValue = currentDictionaryValue+=50;
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                    updatedDictionaryValue);
            }
            else if(positionOfGivenRiderInStage+1==2){
                int updatedDictionaryValue = currentDictionaryValue+=30;
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                    updatedDictionaryValue);
            }
            else if(positionOfGivenRiderInStage+1==3){
                int updatedDictionaryValue = currentDictionaryValue+=20;
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                    updatedDictionaryValue);
            }
            else if(positionOfGivenRiderInStage+1==4){
                int updatedDictionaryValue = currentDictionaryValue+=18;
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                    updatedDictionaryValue);
            }
            else if(positionOfGivenRiderInStage+1==5){
                int updatedDictionaryValue = currentDictionaryValue+=16;
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                    updatedDictionaryValue);
            }
            else if(positionOfGivenRiderInStage+1==6){
```

```
529                 int updatedDictionaryValue = currentDictionaryValue+=14;
530                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
531             }
532             else if(positionOfGivenRiderInStage+1==7){
533                 int updatedDictionaryValue = currentDictionaryValue+=12;
534                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
535             }
536             else if(positionOfGivenRiderInStage+1==8){
537                 int updatedDictionaryValue = currentDictionaryValue+=10;
538                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
539             }
540             else if(positionOfGivenRiderInStage+1==9){
541                 int updatedDictionaryValue = currentDictionaryValue+=8;
542                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
543             }
544             else if(positionOfGivenRiderInStage+1==10){
545                 int updatedDictionaryValue = currentDictionaryValue+=7;
546                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
547             }
548             else if(positionOfGivenRiderInStage+1==11){
549                 int updatedDictionaryValue = currentDictionaryValue+=6;
550                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
551             }
552             else if(positionOfGivenRiderInStage+1==12){
553                 int updatedDictionaryValue = currentDictionaryValue+=5;
554                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
555             }
556             else if(positionOfGivenRiderInStage+1==13){
557                 int updatedDictionaryValue = currentDictionaryValue+=4;
558                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
559             }
560             else if(positionOfGivenRiderInStage+1==14){
561                 int updatedDictionaryValue = currentDictionaryValue+=3;
562                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
563             }
564             else if(positionOfGivenRiderInStage+1==15){
565                 int updatedDictionaryValue = currentDictionaryValue+=2;
566                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
567             }
568         }
569         else if (this.type.equals(StageType.MEDIUM_MOUNTAIN)){
570             if (positionOfGivenRiderInStage+1>15||positionOfGivenRiderInStage+1<1){
571                 break;
572             }else if(positionOfGivenRiderInStage+1==1){
573                 int updatedDictionaryValue = currentDictionaryValue+=30;
```

```
574                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
575             }
576             else if(positionOfGivenRiderInStage+1==2){
577                 int updatedDictionaryValue = currentDictionaryValue+=25;
578                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
579             }
580             else if(positionOfGivenRiderInStage+1==3){
581                 int updatedDictionaryValue = currentDictionaryValue+=22;
582                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
583             }
584             else if(positionOfGivenRiderInStage+1==4){
585                 int updatedDictionaryValue = currentDictionaryValue+=19;
586                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
587             }
588             else if(positionOfGivenRiderInStage+1==5){
589                 int updatedDictionaryValue = currentDictionaryValue+=17;
590                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
591             }
592             else if(positionOfGivenRiderInStage+1==6){
593                 int updatedDictionaryValue = currentDictionaryValue+=15;
594                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
595             }
596             else if(positionOfGivenRiderInStage+1==7){
597                 int updatedDictionaryValue = currentDictionaryValue+=13;
598                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
599             }
600             else if(positionOfGivenRiderInStage+1==8){
601                 int updatedDictionaryValue = currentDictionaryValue+=11;
602                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
603             }
604             else if(positionOfGivenRiderInStage+1==9){
605                 int updatedDictionaryValue = currentDictionaryValue+=9;
606                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
607             }
608             else if(positionOfGivenRiderInStage+1==10){
609                 int updatedDictionaryValue = currentDictionaryValue+=7;
610                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
611             }
612             else if(positionOfGivenRiderInStage+1==11){
613                 int updatedDictionaryValue = currentDictionaryValue+=6;
614                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
615             }
616             else if(positionOfGivenRiderInStage+1==12){
617                 int updatedDictionaryValue = currentDictionaryValue+=5;
```

```
618                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
619             }
620             else if(positionOfGivenRiderInStage+1==13){
621                 int updatedDictionaryValue = currentDictionaryValue+=4;
622                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
623             }
624             else if(positionOfGivenRiderInStage+1==14){
625                 int updatedDictionaryValue = currentDictionaryValue+=3;
626                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
627             }
628             else if(positionOfGivenRiderInStage+1==15){
629                 int updatedDictionaryValue = currentDictionaryValue+=2;
630                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
631             }
632         }
633         else if (this.type.equals(StageType.HIGH_MOUNTAIN)){
634             if (positionOfGivenRiderInStage+1>15||positionOfGivenRiderInStage+1<1){
635                 break;
636             }else if(positionOfGivenRiderInStage+1==1){
637                 int updatedDictionaryValue = currentDictionaryValue+=20;
638                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
639             }
640             else if(positionOfGivenRiderInStage+1==2){
641                 int updatedDictionaryValue = currentDictionaryValue+=17;
642                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
643             }
644             else if(positionOfGivenRiderInStage+1==3){
645                 int updatedDictionaryValue = currentDictionaryValue+=15;
646                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
647             }
648             else if(positionOfGivenRiderInStage+1==4){
649                 int updatedDictionaryValue = currentDictionaryValue+=13;
650                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
651             }
652             else if(positionOfGivenRiderInStage+1==5){
653                 int updatedDictionaryValue = currentDictionaryValue+=11;
654                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
655             }
656             else if(positionOfGivenRiderInStage+1==6){
657                 int updatedDictionaryValue = currentDictionaryValue+=10;
658                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                        updatedDictionaryValue);
659             }
660             else if(positionOfGivenRiderInStage+1==7){
661                 int updatedDictionaryValue = currentDictionaryValue+=9;
662                 riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
```

```
                    updatedDictionaryValue);
            }
            else if(positionOfGivenRiderInStage+1==8){
                int updatedDictionaryValue = currentDictionaryValue+=8;
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                    updatedDictionaryValue);
            }
            else if(positionOfGivenRiderInStage+1==9){
                int updatedDictionaryValue = currentDictionaryValue+=7;
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                    updatedDictionaryValue);
            }
            else if(positionOfGivenRiderInStage+1==10){
                int updatedDictionaryValue = currentDictionaryValue+=6;
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                    updatedDictionaryValue);
            }
            else if(positionOfGivenRiderInStage+1==11){
                int updatedDictionaryValue = currentDictionaryValue+=5;
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                    updatedDictionaryValue);
            }
            else if(positionOfGivenRiderInStage+1==12){
                int updatedDictionaryValue = currentDictionaryValue+=4;
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                    updatedDictionaryValue);
            }
            else if(positionOfGivenRiderInStage+1==13){
                int updatedDictionaryValue = currentDictionaryValue+=3;
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                    updatedDictionaryValue);
            }
            else if(positionOfGivenRiderInStage+1==14){
                int updatedDictionaryValue = currentDictionaryValue+=2;
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                    updatedDictionaryValue);
            }
            else if(positionOfGivenRiderInStage+1==15){
                int updatedDictionaryValue = currentDictionaryValue+=1;
                riderSprintersPointsDictionary.put("Rider "+String.valueOf(givenRiderId),
                    updatedDictionaryValue);
            }
        }
    }
    ArrayList<Integer> arrayListOfSortedRiderIDs = new ArrayList<>();
    for(LocalTime z:arrayListOfRiderElapsedTimes){
        for (Result k:arrayListOfResults){
            if (z.equals(k.getElapsedTime())){
                arrayListOfSortedRiderIDs.add(k.getRiderId());
            }
        }
    }
    ArrayList<LocalTime[]> arrayListOfCheckpointALs = new ArrayList<>();
    for (int y:arrayListOfSortedRiderIDs){
        for (Result i:arrayListOfResults){
```

```
            if ((y)==(i.getRiderId())){
                arrayListOfCheckpointALs.add(i.getCheckpointsWithoutStartAndEnd());
            }
        }
    }
    ArrayList<ArrayList<LocalTime>>arrayListOfSortedCheckpointsForEachRider = new ArrayList<>();
    for (int j=0; j<arrayListOfResults.size(); j++){
        ArrayList<LocalTime> arrayListOfGivenSubElements = new ArrayList<>();
        for (LocalTime[] i:arrayListOfCheckpointALs){
            if (j > i.length|| j==(i.length)){
                break;
            }
            arrayListOfGivenSubElements.add(i[j]);
        }
        Collections.sort(arrayListOfGivenSubElements);
        arrayListOfSortedCheckpointsForEachRider.add(arrayListOfGivenSubElements);
    }
    for (ArrayList<LocalTime> k:arrayListOfSortedCheckpointsForEachRider){
        int indexofelement = arrayListOfSortedCheckpointsForEachRider.indexOf(k);//final
        ArrayList<LocalTime> arrayListOfGivenSubElements = new ArrayList<>();
        for (LocalTime[] i:arrayListOfCheckpointALs){
            arrayListOfGivenSubElements.add((LocalTime)Array.get(i, indexofelement));
        }
        ArrayList<Integer> arrayListOfIndexes = new ArrayList<>();
        for (LocalTime x:k){
            for (LocalTime z:arrayListOfGivenSubElements){
                if (x.equals(z)){
                    int indexofelement2 = arrayListOfGivenSubElements.indexOf(z);//final
                    arrayListOfIndexes.add(indexofelement2);
                }
            }
        }
        outerloop:
        for (int item:arrayListOfIndexes){
            for (int id:arrayListOfSortedRiderIDs){
                if (arrayListOfSortedRiderIDs.indexOf(id) == (item)){
                    int currentDictionaryValue = riderSprintersPointsDictionary.get("Rider
                        "+String.valueOf(id));
                    if
                        (indexofelement==arrayListOfSegments.size()||indexofelement>arrayListOfSegments.size()){
                        break outerloop;
                    }
                    if
                        (arrayListOfSegments.get(indexofelement).getSegmentType().equals(SegmentType.SPRINT)){
                        if (arrayListOfIndexes.indexOf(item)+1>15||arrayListOfIndexes.indexOf(item)+1<1){
                            break;
                        }else if(arrayListOfIndexes.indexOf(item)+1==1){
                            int updatedDictionaryValue = currentDictionaryValue+=20;
                            riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                                updatedDictionaryValue);
                        }
                        else if(arrayListOfIndexes.indexOf(item)+1==2){
                            int updatedDictionaryValue = currentDictionaryValue+=17;
                            riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                                updatedDictionaryValue);
```

```
759                 }
760                 else if(arrayListOfIndexes.indexOf(item)+1==3){
761                     int updatedDictionaryValue = currentDictionaryValue+=15;
762                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
763                 }
764                 else if(arrayListOfIndexes.indexOf(item)+1==4){
765                     int updatedDictionaryValue = currentDictionaryValue+=13;
766                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
767                 }
768                 else if(arrayListOfIndexes.indexOf(item)+1==5){
769                     int updatedDictionaryValue = currentDictionaryValue+=11;
770                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
771                 }
772                 else if(arrayListOfIndexes.indexOf(item)+1==6){
773                     int updatedDictionaryValue = currentDictionaryValue+=10;
774                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
775                 }
776                 else if(arrayListOfIndexes.indexOf(item)+1==7){
777                     int updatedDictionaryValue = currentDictionaryValue+=9;
778                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
779                 }
780                 else if(arrayListOfIndexes.indexOf(item)+1==8){
781                     int updatedDictionaryValue = currentDictionaryValue+=8;
782                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
783                 }
784                 else if(arrayListOfIndexes.indexOf(item)+1==9){
785                     int updatedDictionaryValue = currentDictionaryValue+=7;
786                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
787                 }
788                 else if(arrayListOfIndexes.indexOf(item)+1==10){
789                     int updatedDictionaryValue = currentDictionaryValue+=6;
790                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
791                 }
792                 else if(arrayListOfIndexes.indexOf(item)+1==11){
793                     int updatedDictionaryValue = currentDictionaryValue+=5;
794                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
795                 }
796                 else if(arrayListOfIndexes.indexOf(item)+1==12){
797                     int updatedDictionaryValue = currentDictionaryValue+=4;
798                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
799                 }
800                 else if(arrayListOfIndexes.indexOf(item)+1==13){
801                     int updatedDictionaryValue = currentDictionaryValue+=3;
802                     riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                            updatedDictionaryValue);
```

```
                        }
                        else if(arrayListOfIndexes.indexOf(item)+1==14){
                            int updatedDictionaryValue = currentDictionaryValue+=2;
                            riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                                updatedDictionaryValue);
                        }
                        else if(arrayListOfIndexes.indexOf(item)+1==15){
                            int updatedDictionaryValue = currentDictionaryValue+=1;
                            riderSprintersPointsDictionary.put("Rider "+String.valueOf(id),
                                updatedDictionaryValue);
                        }
                    }
                }
            }
        }
        ArrayList<Integer> arrayListOfPointsCorrespondingToRiderIDsSortedByElapsedTime = new ArrayList<>();
        for (int id:arrayListOfSortedRiderIDs){
            arrayListOfPointsCorrespondingToRiderIDsSortedByElapsedTime.add(riderSprintersPointsDictionary.get("Rider
                "+String.valueOf(id)));
        }
        return arrayListOfPointsCorrespondingToRiderIDsSortedByElapsedTime;
    }

    /**
     * Get the number of points obtained by each rider in a stage.
     * @param allRiders An arraylist of all riders created in the cycling portal for the current race/stage
    * @return @return The ranked list of mountain points each riders received in the stage,
    *        sorted by their finish time.
     *
     *        An empty list if there is no result for the stage.
     */
    public ArrayList<Integer> getRidersMountainPointsInStage(ArrayList<Rider> allRiders) {
        if (arrayListOfResults.isEmpty()){
            return (new ArrayList<Integer>());
        }
        ArrayList<LocalTime> arrayListOfRiderFinishTimes = new ArrayList<>();

        for(Result j:arrayListOfResults){
            arrayListOfRiderFinishTimes.add(j.getFinishTime());
        }

        Collections.sort(arrayListOfRiderFinishTimes);

        ArrayList<Integer> arrayListOfSortedRiderIDs = new ArrayList<>();
        for(LocalTime z:arrayListOfRiderFinishTimes){
            for (Result k:arrayListOfResults){
                if (z.equals(k.getFinishTime())){
                    arrayListOfSortedRiderIDs.add(k.getRiderId());
                }
            }
        }
        ArrayList<LocalTime[]> arrayListOfCheckpointALs = new ArrayList<>();
        for (int y:arrayListOfSortedRiderIDs){
            for (Result i:arrayListOfResults){
```

```
            if ((y)==(i.getRiderId())){
                arrayListOfCheckpointALs.add(i.getCheckpointsWithoutStartAndEnd());
            }
        }
    }
    ArrayList<ArrayList<LocalTime>>arrayListOfSortedCheckpointsForEachRider = new ArrayList<>();
    for (int j=0; j<arrayListOfResults.size(); j++){
        ArrayList<LocalTime> arrayListOfGivenSubElements = new ArrayList<>();
        for (LocalTime[] i:arrayListOfCheckpointALs){
            if (j > i.length|| j==(i.length)){
                break;
            }
            arrayListOfGivenSubElements.add(i[j]);
        }
        Collections.sort(arrayListOfGivenSubElements);
        arrayListOfSortedCheckpointsForEachRider.add(arrayListOfGivenSubElements);
    }
    Hashtable <String, Integer> riderKOMPointsDictionary = this.createRiderKOMPointsDictionary();
    for (ArrayList<LocalTime> k:arrayListOfSortedCheckpointsForEachRider){
        int indexofelement3 = arrayListOfSortedCheckpointsForEachRider.indexOf(k);
        ArrayList<LocalTime> arrayListOfGivenSubElements = new ArrayList<>();
        for (LocalTime[] i:arrayListOfCheckpointALs){
            arrayListOfGivenSubElements.add((LocalTime)Array.get(i, indexofelement3));
        }
        ArrayList<Integer> arrayListOfIndexes2 = new ArrayList<>();
        for (LocalTime x:k){
            for (LocalTime z:arrayListOfGivenSubElements){
                if (x.equals(z)){
                    int indexofelement4 = arrayListOfGivenSubElements.indexOf(z);
                    arrayListOfIndexes2.add(indexofelement4);
                }
            }
        }
        outerloop:
        for (int indexitem:arrayListOfIndexes2){
            for (int id:arrayListOfSortedRiderIDs){
                if (arrayListOfSortedRiderIDs.indexOf(id) == (indexitem)){
                    int currentDictionaryValue = riderKOMPointsDictionary.get("Rider
                        "+String.valueOf(id));
                    if
                        (indexofelement3==arrayListOfSegments.size()||indexofelement3>arrayListOfSegments.size()){
                        break outerloop;
                    }
                    if
                        (arrayListOfSegments.get(indexofelement3).getSegmentType().equals(SegmentType.SPRINT)){
                        break outerloop;
                    }
                    if (arrayListOfSegments.get(indexofelement3).getSegmentType().equals(SegmentType.C4)){
                        if (arrayListOfIndexes2.indexOf(indexitem)+1==1){
                            int updatedDictionaryValue = currentDictionaryValue+=1;
                            riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                                updatedDictionaryValue);
                        }else if(arrayListOfIndexes2.indexOf(indexitem)+1!=1){
                            int updatedDictionaryValue = currentDictionaryValue+=0;
                            riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
```

```
                    updatedDictionaryValue);
906             }
907         }else
                if(arrayListOfSegments.get(indexofelement3).getSegmentType().equals(SegmentType.C3)){
908             if (arrayListOfIndexes2.indexOf(indexitem)+1==1){
909                 int updatedDictionaryValue = currentDictionaryValue+=2;
910                 riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                        updatedDictionaryValue);
911             }else if(arrayListOfIndexes2.indexOf(indexitem)+1==2){
912                 int updatedDictionaryValue = currentDictionaryValue+=1;
913                 riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                        updatedDictionaryValue);
914             }else if(arrayListOfIndexes2.indexOf(indexitem)+1!=1){
915                 int updatedDictionaryValue = currentDictionaryValue+=0;
916                 riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                        updatedDictionaryValue);
917             }
918         }
919         else
                if(arrayListOfSegments.get(indexofelement3).getSegmentType().equals(SegmentType.C2)){
920             if (arrayListOfIndexes2.indexOf(indexitem)+1==1){
921                 int updatedDictionaryValue = currentDictionaryValue+=5;
922                 riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                        updatedDictionaryValue);
923             }else if(arrayListOfIndexes2.indexOf(indexitem)+1==2){
924                 int updatedDictionaryValue = currentDictionaryValue+=3;
925                 riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                        updatedDictionaryValue);
926             }else if(arrayListOfIndexes2.indexOf(indexitem)+1==3){
927                 int updatedDictionaryValue = currentDictionaryValue+=2;
928                 riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                        updatedDictionaryValue);
929             }else if(arrayListOfIndexes2.indexOf(indexitem)+1==4){
930                 int updatedDictionaryValue = currentDictionaryValue+=1;
931                 riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                        updatedDictionaryValue);
932             }else if(arrayListOfIndexes2.indexOf(indexitem)+1!=1){
933                 int updatedDictionaryValue = currentDictionaryValue+=0;
934                 riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                        updatedDictionaryValue);
935             }
936         }else
                if(arrayListOfSegments.get(indexofelement3).getSegmentType().equals(SegmentType.C1)){
937             if (arrayListOfIndexes2.indexOf(indexitem)+1==1){
938                 int updatedDictionaryValue = currentDictionaryValue+=10;
939                 riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                        updatedDictionaryValue);
940             }else if(arrayListOfIndexes2.indexOf(indexitem)+1==2){
941                 int updatedDictionaryValue = currentDictionaryValue+=8;
942                 riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                        updatedDictionaryValue);
943             }else if(arrayListOfIndexes2.indexOf(indexitem)+1==3){
944                 int updatedDictionaryValue = currentDictionaryValue+=6;
945                 riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                        updatedDictionaryValue);
```

```
946                         }else if(arrayListOfIndexes2.indexOf(indexitem)+1==4){
947                             int updatedDictionaryValue = currentDictionaryValue+=4;
948                             riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                                    updatedDictionaryValue);
949                         }else if(arrayListOfIndexes2.indexOf(indexitem)+1==5){
950                             int updatedDictionaryValue = currentDictionaryValue+=2;
951                             riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                                    updatedDictionaryValue);
952                         }else if(arrayListOfIndexes2.indexOf(indexitem)+1==6){
953                             int updatedDictionaryValue = currentDictionaryValue+=1;
954                             riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                                    updatedDictionaryValue);
955                         }else if(arrayListOfIndexes2.indexOf(indexitem)+1!=1){
956                             int updatedDictionaryValue = currentDictionaryValue+=0;
957                             riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                                    updatedDictionaryValue);
958                         }
959                     }
960                     else if
                         (arrayListOfSegments.get(indexofelement3).getSegmentType().equals(SegmentType.HC)){
961                         if (arrayListOfIndexes2.indexOf(indexitem)+1==1){
962                             int updatedDictionaryValue = currentDictionaryValue+=20;
963                             riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                                    updatedDictionaryValue);
964                         }else if(arrayListOfIndexes2.indexOf(indexitem)+1==2){
965                             int updatedDictionaryValue = currentDictionaryValue+=15;
966                             riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                                    updatedDictionaryValue);
967                         }else if(arrayListOfIndexes2.indexOf(indexitem)+1==3){
968                             int updatedDictionaryValue = currentDictionaryValue+=12;
969                             riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                                    updatedDictionaryValue);
970                         }else if(arrayListOfIndexes2.indexOf(indexitem)+1==4){
971                             int updatedDictionaryValue = currentDictionaryValue+=10;
972                             riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                                    updatedDictionaryValue);
973                         }else if(arrayListOfIndexes2.indexOf(indexitem)+1==5){
974                             int updatedDictionaryValue = currentDictionaryValue+=8;
975                             riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                                    updatedDictionaryValue);
976                         }else if(arrayListOfIndexes2.indexOf(indexitem)+1==6){
977                             int updatedDictionaryValue = currentDictionaryValue+=6;
978                             riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                                    updatedDictionaryValue);
979                         }else if(arrayListOfIndexes2.indexOf(indexitem)+1==7){
980                             int updatedDictionaryValue = currentDictionaryValue+=4;
981                             riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                                    updatedDictionaryValue);
982                         }else if(arrayListOfIndexes2.indexOf(indexitem)+1==8){
983                             int updatedDictionaryValue = currentDictionaryValue+=2;
984                             riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
                                    updatedDictionaryValue);
985                         }else if(arrayListOfIndexes2.indexOf(indexitem)+1!=1){
986                             int updatedDictionaryValue = currentDictionaryValue+=0;
987                             riderKOMPointsDictionary.put("Rider "+String.valueOf(id),
```

```
                                updatedDictionaryValue);
                        }
                    }
                }
            }
        }
        ArrayList<Integer> arrayListOfMountainPointsCorrespondingToRiderIDsSortedByFinishTime = new
             ArrayList<>();
        for (int id:arrayListOfSortedRiderIDs){
            arrayListOfMountainPointsCorrespondingToRiderIDsSortedByFinishTime.add(riderKOMPointsDictionary.get("Rider
                "+String.valueOf(id)));
        }
        return arrayListOfMountainPointsCorrespondingToRiderIDsSortedByFinishTime;
    }
}
```

4 Segment.java

```
package cycling;

import java.io.Serializable;

/**
 * The Segment class is a parent class for CategorizedClimb and IntermediateSprint.
 */
abstract class Segment implements Serializable {
    public static int segmentCounter = 0;
    public int segmentId;

    /**
     * Constructor for the Segment class.
     */
    public Segment() {
        segmentCounter += 1;
        segmentId = segmentCounter;
    }

    /**
     * Gets the ID for the segment.
     * @return The segment ID.
     */
    public int getSegmentId() {
        return segmentId;
    }

    /**
     * Gets the type of the segment.
     * @return
     */
    public SegmentType getSegmentType() {
        return null;
    }
```

```
35
36      /**
37       * Gets the kilometre location where the segment finishes within the stage.
38       * @return
39       */
40      public Double getLocation() {
41          return null;
42      }
43  }
```

5 CategorizedClimb.java

```
1   package cycling;
2
3   /**
4    * The CategorizedClimb class is a child class of Segment and stores all the data about a
5    *  categorized climb segment in a stage.
6    */
7   public class CategorizedClimb extends Segment {
8       private Double location;
9       private SegmentType type;
10      private Double averageGradient;
11      private Double length;
12      private int stageId;
13
14      /**
15       * Constructor for the CategorizedClimb class.
16       * @param stageId The ID if the stage to which the climb segment is being added.
17       * @param location The kilometre location where the climb finishes within the stage.
18       * @param type The category of the climb - {@link SegmentType#C4},
19       * {@link SegmentType#C3}, {@link SegmentType#C2},
20       * {@link SegmentType#C1}, or {@link SegmentType#HC}.
21       * @param averageGradient The average gradient for the climb.
22       * @param length The length of the climb in kilometres.
23       */
24      public CategorizedClimb(int stageId, Double location, SegmentType type,
25                              Double averageGradient, Double length) {
26          this.location = location;
27          this.type = type;
28          this.averageGradient = averageGradient;
29          this.length = length;
30          this.stageId = stageId;
31      }
32
33      /**
34       * Gets the type of segment for the categorized climb.
35       * @return The segment type for the categorized climb.
36       */
37      public SegmentType getSegmentType() {
38          return type;
39      }
40
41      /**
42       * Gets the kilometre location where the climb finishes within the stage.
```

```
     * @return The kilometre location where the climb finishes within the stage.
     */
    public Double getLocation() {
        return location;
    }
}
```

6 IntermediateSprint.java

```
package cycling;

/**
 * The IntermediateSprint class is a child class of segment and stores all the data about an
 * intermediate sprint segment in a stage.
 */
public class IntermediateSprint extends Segment {
    private double location;
    private SegmentType type = SegmentType.SPRINT;
    private int stageId;

    /**
     * Constructor for the IntermediateSprint class.
     * @param stageId The ID of the stage to which the intermediate sprint segment is being added.
     * @param location The kilometre location where the intermediate sprint finishes
     *       within the stage.
     * @param type The type of the segment.
     */
    public IntermediateSprint(int stageId, double location) {
        this.location = location;
        this.stageId = stageId;
    }

    /**
     * Gets the type of the segment.
     * @return The segment type.
     */
    public SegmentType getSegmentType() {
        return type;
    }

    /**
     * Gets the kilometre location where the intermediate sprint finishes within the stage.
     * @return The kilometre location where the intermediate sprint finishes within the stage.
     */
    public Double getLocation() {
        return location;
    }
}
```

7 Team.java

```
package cycling;

import java.io.Serializable;
```

```
4  import java.util.ArrayList;
5
6  /**
7   * The Team class stores all the informaton related to a team,
8   *  including which riders are a part of the team.
9   */
10 public class Team implements Serializable {
11     public static int teamCounter = 0;
12     private int teamId;
13     private String name;
14     private String description;
15     ArrayList<Rider> arrayListOfRiders = new ArrayList<>();
16
17     /**
18      * Constructor for the team class.
19      * @param name The name of the team.
20      * @param description A description of the team.
21      */
22     public Team(String name, String description) {
23         this.name = name;
24         this.description = description;
25         teamCounter += 1;
26         teamId = teamCounter;
27     }
28
29     /**
30      * Gets the ID for the team.
31      * @return The team ID.
32      */
33     public int getTeamId() {
34         return teamId;
35     }
36
37     /**
38      * Gets the name of the team.
39      * @return The team name.
40      */
41     public String getName() {
42         return name;
43     }
44
45     /**
46      * Gets the description of the team.
47      * @return The description of the team.
48      */
49     public String getDescription() {
50         return description;
51     }
52
53     /**
54      * Creates a new rider and adds it to a list of riders that belong to the team.
55      * @param teamId The ID of the team the rider belongs to.
56      * @param name The name of the rider.
57      * @param yearOfBirth The year of birth for the rider.
58      * @return The ID created for the rider
```

```
     */
    public int createRider(int teamId, String name, int yearOfBirth) {
        Rider newRider = new Rider(teamId, name, yearOfBirth);
        arrayListOfRiders.add(newRider);
        return newRider.getRiderId();
    }

    /**
     * Removes a rider and removes it from the team.
     * @param riderId The ID of the rider
     */
    public void removeRider(int riderId) {
        for (Rider i:arrayListOfRiders) {
            if ((i.getRiderId()) == (riderId)) {
                arrayListOfRiders.remove(i);
            }
        }
    }

    /**
     * Gets all the riders that belong to the team.
     * @return A list of Riders that belong to the team.
     */
    public ArrayList<Integer> getTeamRiders() {
        ArrayList<Integer> arrayListOfTeamRiderIDs = new ArrayList<>();
        for (Rider rider:arrayListOfRiders) {
            arrayListOfTeamRiderIDs.add(rider.getRiderId());
        }
        return arrayListOfTeamRiderIDs;
    }
}
```

8 Rider.java

```
package cycling;

import java.io.Serializable;

/**
 * The Rider class stores all the data related to the rider.
 */
public class Rider implements Serializable {
    public static int riderCounter = 0;
    private int riderId;
    private int teamId;
    private String name;
    private int yearOfBirth;

    /**
     * Constructor for the Rider class.
     * @param teamId The ID of the rider's team.
     * @param name The name of the rider.
     * @param yearOfBirth The year of birth of the rider.
     * @see cycling.team
```

```
     */
    public Rider(int teamId, String name, int yearOfBirth) {
        this.teamId = teamId;
        this.name = name;
        this.yearOfBirth = yearOfBirth;
        riderCounter ++;
        riderId = riderCounter;
    }

    /**
     * Gets the team ID of the team the rider belongs to.
     * @return The team ID.
     */
    public int getTeamId() {
        return teamId;
    }

    /**
     * Gets the ID for the rider.
     * @return The rider ID.
     */
    public int getRiderId() {
        return riderId;
    }

    /**
     * Gets the name of the rider.
     * @return The name of the rider.
     */
    public String getName() {
        return name;
    }

    /**
     * Gets the year of birth of the rider.
     * @return The year of birth of the rider.
     */
    public int getYearOfBirth() {
        return yearOfBirth;
    }
}
```

9 Result.java

```
package cycling;

import java.io.Serializable;
import java.util.ArrayList;
import java.time.LocalTime;
import java.time.temporal.ChronoUnit;
import java.time.Duration;
import java.util.Hashtable;
import cycling.StageType;
import java.util.Collections;
```

```
11
12  /**
13   * The Result class holds all the information for results, points, and works out the elapsed time and
         adjusted elapsed times.
14   */
15  public class Result implements Serializable {
16      private int stageId;
17      private int riderId;
18      private LocalTime[] checkpoints;
19
20      /**
21       * Constructor for the result class.
22       * @param stageId The ID of the stage the result refers to.
23       * @param riderId The ID of the rider the result is for.
24       * @param checkpoints An array of times at which the rider reached each of the segments of the stage,
             including the start time and the finish line.
25       */
26      public Result(int stageId, int riderId, LocalTime... checkpoints) {
27          this.stageId = stageId;
28          this.riderId = riderId;
29          this.checkpoints = checkpoints;
30      }
31
32      /**
33       * Gets the stage ID for the result.
34       * @return The stage ID.
35       */
36      public int getStageId(){
37          return stageId;
38      }
39
40      /**
41       * Gets the rider ID for the result.
42       * @return The rider ID.
43       */
44      public int getRiderId(){
45          return riderId;
46      }
47
48      /**
49       * Gets the array of times at which the rider reached each of the segments of the stage with the start
             time and the finish line, for the result.
50       * @return An array of times.
51       */
52      public LocalTime[] getCheckpoints() {
53          return checkpoints;
54      }
55
56      /**
57       * Gets the finish time from the array of times the rider started, finished and reached the segments of
             the stage.
58       * @return The finish time.
59       */
60      public LocalTime getFinishTime() {
61          return checkpoints[checkpoints.length -1];
```

```
62      }
63
64      /**
65       * Gets all the times at which the rider reached the segments of the stage, not including the start and
             finish time for a result.
66       * @return An array of times which the rider reached the segments of the stage.
67       */
68      public LocalTime[] getCheckpointsWithoutStartAndEnd() {
69          ArrayList<LocalTime> arrayListOfCheckpointsWithoutStartAndEnd = new ArrayList<LocalTime>();
70          for (int i = 1; i < checkpoints.length; i++) {
71              arrayListOfCheckpointsWithoutStartAndEnd.add(checkpoints[i]);
72          }
73          LocalTime[] checkpointsWithoutStartAndEnd = arrayListOfCheckpointsWithoutStartAndEnd.toArray(new
                 LocalTime[arrayListOfCheckpointsWithoutStartAndEnd.size()]);
74          return checkpointsWithoutStartAndEnd;
75      }
76
77      /**
78       * Gets the amount of time between the start time and finish time of a result.
79       * @return the elapsed time for a result
80       */
81      public LocalTime getElapsedTime(){
82          Duration durationBetween = Duration.between(checkpoints[0], checkpoints[checkpoints.length-1]);
83          long elapsedTimeInNanoseconds = durationBetween.toNanos();
84          LocalTime elapsedTime = LocalTime.ofNanoOfDay(elapsedTimeInNanoseconds);
85          return elapsedTime;
86      }
87
88      /**
89       * Gets the adjusted elapsed time for a rider in a stage, checking if another rider finished within 1
             second of another and adjusting the time to be the smallest of both.
90       * @param arrayListOfResults All rider results for a stage.
91       * @return The adjusted elapsed time for the rider in a stage. Returns an empty array if there is no
             result for the rider in the stage.
92       */
93      public LocalTime getAdjustedElapsedTime(ArrayList<Result> arrayListOfResults){
94          ArrayList<LocalTime> arrayListOfElapsedTimesForStage = new ArrayList<LocalTime>();
95          ArrayList<Integer> arrayListOfRiderIds = new ArrayList<Integer>();
96          for (int counter = 0; counter < arrayListOfResults.size(); counter++) {
97              arrayListOfElapsedTimesForStage.add((arrayListOfResults.get(counter)).getElapsedTime());
98              arrayListOfRiderIds.add((arrayListOfResults.get(counter)).getRiderId());
99          }
100         ArrayList<Integer> arrayListOfIndexes = new ArrayList<>();
101         ArrayList<Integer> arrayListOfOrderedRiderIDs = new ArrayList<>();
102
103         ArrayList<LocalTime> arrayListOfElapsedTimesForStageOld = new
                 ArrayList<>(arrayListOfElapsedTimesForStage);
104         Collections.sort(arrayListOfElapsedTimesForStage);
105         for (LocalTime elapsedT: arrayListOfElapsedTimesForStage){
106             arrayListOfIndexes.add(arrayListOfElapsedTimesForStageOld.indexOf(elapsedT));
107         }for (int index:arrayListOfIndexes){
108             arrayListOfOrderedRiderIDs.add(arrayListOfRiderIds.get(index));
109         }
110
111         for (int i = 0; i < (arrayListOfElapsedTimesForStage.size()-1); i++) {
```

```
112             if (arrayListOfElapsedTimesForStage.get(i).until(arrayListOfElapsedTimesForStage.get(i+1),
                    ChronoUnit.SECONDS)<60){
113             //if
                    ((arrayListOfElapsedTimesForStage.get(i).plusNanos(1000000000).compareTo(arrayListOfElapsedTimesForStag
                    { //If the time is less than 1000000000 nanoseconds behind the next time
114                 arrayListOfElapsedTimesForStage.set(i+1, arrayListOfElapsedTimesForStage.get(i)); //Adjust
                        time
115             }
116         }
117
118
119         for (int q = 0; q < arrayListOfElapsedTimesForStage.size(); q++) {
120             //if ((arrayListOfRiderIds.get(q)).equals(this.riderId)) {
121             if ((arrayListOfOrderedRiderIDs.get(q)).equals(this.riderId)) {
122                 //int idindex = arrayListOfOrderedRiderIDs.indexOf(q);
123                 LocalTime adjustedElapsedTimeForRider = arrayListOfElapsedTimesForStage.get(q);
124                 return adjustedElapsedTimeForRider;
125             }
126         }return LocalTime.of(0,0,0,0);
127     }
128 }
```