

# CyclingPortal Printout

123456789 & 987654321

## Contents

1	CategorizedClimb.java	2
2	CyclingPortal.java	2
3	IntermediateSprint.java	12
4	Race.java	12
5	RaceResult.java	14
6	Rider.java	15
7	Segment.java	16
8	SegmentResult.java	19
9	Stage.java	20
10	StageResult.java	24
11	Team.java	25

## 1 CategorizedClimb.java

```
1 package cycling;
2
3 public class CategorizedClimb extends Segment {
4     private final Double averageGradient;
5     private final Double length;
6
7     public CategorizedClimb(
8         Stage stage, Double location, SegmentType type, Double averageGradient, Double
9         ↪ length)
10        throws InvalidLocationException, InvalidStageStateException,
11        ↪ InvalidStageTypeException {
12        super(stage, type, location);
13        this.averageGradient = averageGradient;
14        this.length = length;
15    }
16 }
```

## 2 CyclingPortal.java

```
1 package cycling;
2
3 import java.io.*;
4 import java.time.LocalDateTime;
5 import java.time.LocalTime;
6 import java.util.ArrayList;
7 import java.util.List;
8
9 // TODO:
10 //     - Asserts !!!!
11 //     - Code Formatting
12 //     - Documentation/Comments
13 //     - Testing
14 //     - test all removes are removing everything associated with that thing
15 //     - each function public/private/protected/default
16 //     - Optimise results?
17
18 public class CyclingPortal implements CyclingPortalInterface {
19
20     private ArrayList<Team> teams = new ArrayList<>();
21     private ArrayList<Rider> riders = new ArrayList<>();
22     private ArrayList<Race> races = new ArrayList<>();
23     private ArrayList<Stage> stages = new ArrayList<>();
24     private ArrayList<Segment> segments = new ArrayList<>();
25
26     private record SavedCyclingPortal(
27         ArrayList<Team> teams,
28         ArrayList<Rider> riders,
29         ArrayList<Race> races,
30         ArrayList<Stage> stages,
31         ArrayList<Segment> segments,
32         int teamIdCount,
33         int riderIdCount,
34         int raceIdCount,
35         int stageIdCount,
36         int segmentIdCount) {}
37 }
```

```
37
38 public static boolean containsWhitespace(String str) {
39     for (int i = 0; i < str.length(); ++i) {
40         if (Character.isWhitespace(str.charAt(i))) {
41             return true;
42         }
43     }
44     return false;
45 }
46
47 public Team getTeamById(int ID) throws IDNotRecognisedException {
48     for (Team team : teams) {
49         if (team.getId() == ID) {
50             return team;
51         }
52     }
53     throw new IDNotRecognisedException("Team ID not found.");
54 }
55
56 public Rider getRiderById(int ID) throws IDNotRecognisedException {
57     for (Rider rider : riders) {
58         if (rider.getId() == ID) {
59             return rider;
60         }
61     }
62     throw new IDNotRecognisedException("Racer ID not found.");
63 }
64
65 public Race getRaceById(int ID) throws IDNotRecognisedException {
66     for (Race race : races) {
67         if (race.getId() == ID) {
68             return race;
69         }
70     }
71     throw new IDNotRecognisedException("Race ID not found.");
72 }
73
74 public Stage getStageById(int ID) throws IDNotRecognisedException {
75     for (Stage stage : stages) {
76         if (stage.getId() == ID) {
77             return stage;
78         }
79     }
80     throw new IDNotRecognisedException("Stage ID not found.");
81 }
82
83 public Segment getSegmentById(int ID) throws IDNotRecognisedException {
84     for (Segment segment : segments) {
85         if (segment.getId() == ID) {
86             return segment;
87         }
88     }
89     throw new IDNotRecognisedException("Segment ID not found.");
90 }
91
92 public void removeRiderResults(Rider rider) {
93     for (Race race : races) {
94         race.removeRiderResults(rider);
```

```
95     }
96     for (Stage stage : stages) {
97         stage.removeRiderResults(rider);
98     }
99     for (Segment segment : segments) {
100         segment.removeRiderResults(rider);
101     }
102 }
103
104 @Override
105 public int[] getRaceIds() {
106     int[] raceIDs = new int[races.size()];
107     for (int i = 0; i < races.size(); i++) {
108         Race race = races.get(i);
109         raceIDs[i] = race.getId();
110     }
111     return raceIDs;
112 }
113
114 @Override
115 public int createRace(String name, String description)
116     throws IllegalArgumentException, InvalidNameException {
117     for (Race race : races) {
118         if (race.getName().equals(name)) {
119             throw new IllegalArgumentException("A Race with the name " + name + " already
120                 ↪ exists.");
121         }
122     }
123     Race race = new Race(name, description);
124     races.add(race);
125     return race.getId();
126 }
127
128 @Override
129 public String viewRaceDetails(int raceId) throws IDNotRecognisedException {
130     Race race = getRaceById(raceId);
131     return race.getDetails();
132 }
133
134 @Override
135 public void removeRaceById(int raceId) throws IDNotRecognisedException {
136     Race race = getRaceById(raceId);
137     for (final Stage stage : race.getStages()) {
138         stages.remove(stage);
139     }
140     races.remove(race);
141 }
142
143 @Override
144 public int getNumberOfStages(int raceId) throws IDNotRecognisedException {
145     Race race = getRaceById(raceId);
146     return race.getStages().size();
147 }
148
149 @Override
150 public int addStageToRace(
151     int raceId,
152     String stageName,
```

```

152     String description,
153     double length,
154     LocalDateTime startTime,
155     StageType type)
156     throws IDNotRecognisedException, IllegalNameException, InvalidNameException,
157            InvalidLengthException {
158     Race race = getRaceById(raceId);
159     for (final Stage stage : stages) {
160         if (stage.getName().equals(stageName)) {
161             throw new IllegalNameException("A stage with the name " + stageName + "
162                 ↳ already exists.");
163         }
164     }
165     Stage stage = new Stage(race, stageName, description, length, startTime, type);
166     stages.add(stage);
167     race.addStage(stage);
168     return stage.getId();
169 }
170
171 @Override
172 public int[] getRaceStages(int raceId) throws IDNotRecognisedException {
173     Race race = getRaceById(raceId);
174     ArrayList<Stage> raceStages = race.getStages();
175     int[] raceStagesId = new int[raceStages.size()];
176     for (int i = 0; i < raceStages.size(); i++) {
177         Stage stage = race.getStages().get(i);
178         raceStagesId[i] = stage.getId();
179     }
180     return raceStagesId;
181 }
182
183 @Override
184 public double getStageLength(int stageId) throws IDNotRecognisedException {
185     Stage stage = getStageById(stageId);
186     return stage.getLength();
187 }
188
189 @Override
190 public void removeStageById(int stageId) throws IDNotRecognisedException {
191     Stage stage = getStageById(stageId);
192     Race race = stage.getRace();
193     race.removeStage(stage);
194     stages.remove(stage);
195 }
196
197 @Override
198 public int addCategorizedClimbToStage(
199     int stageId, Double location, SegmentType type, Double averageGradient, Double
200     ↳ length)
201     throws IDNotRecognisedException, InvalidLocationException,
202            InvalidStageStateException,
203            InvalidStageTypeException {
204     Stage stage = getStageById(stageId);
205     CategorizedClimb climb = new CategorizedClimb(stage, location, type,
206         ↳ averageGradient, length);
207     segments.add(climb);
208     stage.addSegment(climb);
209     return climb.getId();

```

```
206     }
207
208     @Override
209     public int addIntermediateSprintToStage(int stageId, double location)
210         throws IDNotRecognisedException, InvalidLocationException,
211             InvalidStageStateException,
212             InvalidStageTypeException {
213         Stage stage = getStageById(stageId);
214         IntermediateSprint sprint = new IntermediateSprint(stage, location);
215         segments.add(sprint);
216         stage.addSegment(sprint);
217         return sprint.getId();
218     }
219
220     @Override
221     public void removeSegment(int segmentId)
222         throws IDNotRecognisedException, InvalidStageStateException {
223         Segment segment = getSegmentById(segmentId);
224         Stage stage = segment.getStage();
225         stage.removeSegment(segment);
226         segments.remove(segment);
227     }
228
229     @Override
230     public void concludeStagePreparation(int stageId)
231         throws IDNotRecognisedException, InvalidStageStateException {
232         Stage stage = getStageById(stageId);
233         stage.concludePreparation();
234     }
235
236     @Override
237     public int[] getStageSegments(int stageId) throws IDNotRecognisedException {
238         Stage stage = getStageById(stageId);
239         ArrayList<Segment> stageSegments = stage.getSegments();
240         int[] stageSegmentsId = new int[stageSegments.size()];
241         for (int i = 0; i < stageSegments.size(); i++) {
242             Segment segment = stageSegments.get(i);
243             stageSegmentsId[i] = segment.getId();
244         }
245         return stageSegmentsId;
246     }
247
248     @Override
249     public int createTeam(String name, String description)
250         throws IllegalNameException, InvalidNameException {
251         for (final Team team : teams) {
252             if (team.getName().equals(name)) {
253                 throw new IllegalNameException("A Team with the name " + name + " already
254                     exists.");
255             }
256         }
257         Team team = new Team(name, description);
258         teams.add(team);
259         return team.getId();
260     }
261
262     @Override
263     public void removeTeam(int teamId) throws IDNotRecognisedException {
```

```
262     Team team = getTeamById(teamId);
263     for (final Rider rider : team.getRiders()) {
264         removeRiderResults(rider);
265         riders.remove(rider);
266     }
267     teams.remove(team);
268 }
269
270 @Override
271 public int[] getTeams() {
272     int[] teamIDs = new int[teams.size()];
273     for (int i = 0; i < teams.size(); i++) {
274         Team team = teams.get(i);
275         teamIDs[i] = team.getId();
276     }
277     return teamIDs;
278 }
279
280 @Override
281 public int[] getTeamRiders(int teamId) throws IDNotRecognisedException {
282     Team team = getTeamById(teamId);
283     ArrayList<Rider> teamRiders = team.getRiders();
284     int[] teamRiderIds = new int[teamRiders.size()];
285     for (int i = 0; i < teamRiderIds.length; i++) {
286         teamRiderIds[i] = teamRiders.get(i).getId();
287     }
288     return teamRiderIds;
289 }
290
291 @Override
292 public int createRider(int teamID, String name, int yearOfBirth)
293     throws IDNotRecognisedException, IllegalArgumentException {
294     Team team = getTeamById(teamID);
295     Rider rider = new Rider(team, name, yearOfBirth);
296     team.addRider(rider);
297     riders.add(rider);
298     return rider.getId();
299 }
300
301 @Override
302 public void removeRider(int riderId) throws IDNotRecognisedException {
303     Rider rider = getRiderById(riderId);
304     removeRiderResults(rider);
305     rider.getTeam().removeRider(rider);
306     riders.remove(rider);
307 }
308
309 @Override
310 public void registerRiderResultsInStage(int stageId, int riderId, LocalTime...
311     ↪ checkpoints)
312     throws IDNotRecognisedException, DuplicatedResultException,
313     ↪ InvalidCheckpointsException,
314     InvalidStageStateException {
315     Stage stage = getStageById(stageId);
316     Rider rider = getRiderById(riderId);
317     stage.registerResult(rider, checkpoints);
318 }
```

```
318  @Override
319  public LocalTime[] getRiderResultsInStage(int stageId, int riderId)
320      throws IDNotRecognisedException {
321      Stage stage = getStageById(stageId);
322      Rider rider = getRiderById(riderId);
323      StageResult result = stage.getRiderResult(rider);
324
325      if (result == null) {
326          return new LocalTime[] {};
327      } else {
328          LocalTime[] checkpoints = result.getCheckpoints();
329          LocalTime[] resultsInStage = new LocalTime[checkpoints.length + 1];
330          LocalTime elapsedTime = LocalTime.MIDNIGHT.plus(result.getElapsedTime());
331          for (int i = 0; i <= resultsInStage.length; i++) {
332              if (i == resultsInStage.length) {
333                  resultsInStage[i] = elapsedTime;
334              } else {
335                  resultsInStage[i] = checkpoints[i];
336              }
337          }
338          return resultsInStage;
339      }
340  }
341
342  @Override
343  public LocalTime getRiderAdjustedElapsedTimeInStage(int stageId, int riderId)
344      throws IDNotRecognisedException {
345      Stage stage = getStageById(stageId);
346      Rider rider = getRiderById(riderId);
347      StageResult result = stage.getRiderResult(rider);
348      if (result == null) {
349          return null;
350      } else {
351          return result.getAdjustedElapsedLocalTime();
352      }
353  }
354
355  @Override
356  public void deleteRiderResultsInStage(int stageId, int riderId) throws
357      ↪ IDNotRecognisedException {
358      Stage stage = getStageById(stageId);
359      Rider rider = getRiderById(riderId);
360      stage.removeRiderResults(rider);
361  }
362
363  @Override
364  public int[] getRidersRankInStage(int stageId) throws IDNotRecognisedException {
365      Stage stage = getStageById(stageId);
366      List<Rider> riders = stage.getRidersByElapsedTime();
367      int[] riderIds = new int[riders.size()];
368      for (int i = 0; i < riders.size(); i++) {
369          riderIds[i] = riders.get(i).getId();
370      }
371      return riderIds;
372  }
373
374  @Override
375  public LocalTime[] getRankedAdjustedElapsedTimesInStage(int stageId)
```



```
375         throws IDNotRecognisedException {
376     Stage stage = getStageById(stageId);
377     List<Rider> riders = stage.getRidersByElapsedTime();
378     LocalTime[] riderAETs = new LocalTime[riders.size()];
379     for (int i = 0; i < riders.size(); i++) {
380         Rider rider = riders.get(i);
381         riderAETs[i] = stage.getRiderResult(rider).getAdjustedElapsedLocalTime();
382     }
383     return riderAETs;
384 }
385
386 @Override
387 public int[] getRidersPointsInStage(int stageId) throws IDNotRecognisedException {
388     Stage stage = getStageById(stageId);
389     List<Rider> riders = stage.getRidersByElapsedTime();
390     int[] riderSprinterPoints = new int[riders.size()];
391     for (int i = 0; i < riders.size(); i++) {
392         Rider rider = riders.get(i);
393         riderSprinterPoints[i] = stage.getRiderResult(rider).getSprintersPoints();
394     }
395     return riderSprinterPoints;
396 }
397
398 @Override
399 public int[] getRidersMountainPointsInStage(int stageId) throws
↳ IDNotRecognisedException {
400     Stage stage = getStageById(stageId);
401     List<Rider> riders = stage.getRidersByElapsedTime();
402     int[] riderMountainPoints = new int[riders.size()];
403     for (int i = 0; i < riders.size(); i++) {
404         Rider rider = riders.get(i);
405         riderMountainPoints[i] = stage.getRiderResult(rider).getMountainPoints();
406     }
407     return riderMountainPoints;
408 }
409
410 @Override
411 public void eraseCyclingPortal() {
412     teams = new ArrayList<>();
413     riders = new ArrayList<>();
414     races = new ArrayList<>();
415     stages = new ArrayList<>();
416     segments = new ArrayList<>();
417     Rider.resetIdCounter();
418     Team.resetIdCounter();
419     Race.resetIdCounter();
420     Stage.resetIdCounter();
421     Segment.resetIdCounter();
422 }
423
424 @Override
425 public void saveCyclingPortal(String filename) throws IOException {
426     FileOutputStream file = new FileOutputStream(filename);
427     ObjectOutputStream output = new ObjectOutputStream(file);
428     SavedCyclingPortal savedCyclingPortal =
429         new SavedCyclingPortal(
430         teams,
431         riders,
```

```
432         races,
433         stages,
434         segments,
435         Team.getIdCounter(),
436         Rider.getIdCounter(),
437         Race.getIdCounter(),
438         Stage.getIdCounter(),
439         Segment.getIdCounter());
440     output.writeObject(savedCyclingPortal);
441     output.close();
442     file.close();
443 }
444
445 @Override
446 public void loadCyclingPortal(String filename) throws IOException,
↳ ClassNotFoundException {
447     eraseCyclingPortal();
448     FileInputStream file = new FileInputStream(filename);
449     ObjectInputStream input = new ObjectInputStream(file);
450
451     SavedCyclingPortal savedCyclingPortal = (SavedCyclingPortal) input.readObject();
452     teams = savedCyclingPortal.teams;
453     riders = savedCyclingPortal.riders;
454     races = savedCyclingPortal.races;
455     stages = savedCyclingPortal.stages;
456     segments = savedCyclingPortal.segments;
457
458     Team.setIdCounter(savedCyclingPortal.teamIdCount);
459     Rider.setIdCounter(savedCyclingPortal.riderIdCount);
460     Race.setIdCounter(savedCyclingPortal.raceIdCount);
461     Stage.setIdCounter(savedCyclingPortal.stageIdCount);
462     Segment.setIdCounter(savedCyclingPortal.segmentIdCount);
463
464     input.close();
465     file.close();
466 }
467
468 @Override
469 public void removeRaceByName(String name) throws NameNotRecognisedException {
470     for (final Race race : races) {
471         if (race.getName().equals(name)) {
472             races.remove(race);
473             return;
474         }
475     }
476     throw new NameNotRecognisedException("Race name is not in the system.");
477 }
478
479 @Override
480 public int[] getRidersGeneralClassificationRank(int raceId) throws
↳ IDNotRecognisedException {
481     Race race = getRaceById(raceId);
482     List<Rider> riders = race.getRidersByAdjustedElapsedTime();
483     int[] riderIds = new int[riders.size()];
484     for (int i = 0; i < riders.size(); i++) {
485         riderIds[i] = riders.get(i).getId();
486     }
487     return riderIds;
```

```
488     }
489
490     @Override
491     public LocalTime[] getGeneralClassificationTimesInRace(int raceId)
492         throws IDNotRecognisedException {
493         Race race = getRaceById(raceId);
494         List<Rider> riders = race.getRidersByAdjustedElapsedTime();
495         LocalTime[] riderTimes = new LocalTime[riders.size()];
496         for (int i = 0; i < riders.size(); i++) {
497             riderTimes[i] =
498                 ↪ race.getRiderResults(riders.get(i)).getCumulativeAdjustedElapsedLocalTime();
499         }
500         return riderTimes;
501     }
502
503     @Override
504     public int[] getRidersPointsInRace(int raceId) throws IDNotRecognisedException {
505         Race race = getRaceById(raceId);
506         List<Rider> riders = race.getRidersByAdjustedElapsedTime();
507         int[] riderIds = new int[riders.size()];
508         for (int i = 0; i < riders.size(); i++) {
509             riderIds[i] =
510                 ↪ race.getRiderResults(riders.get(i)).getCumulativeSprintersPoints();
511         }
512         return riderIds;
513     }
514
515     @Override
516     public int[] getRidersMountainPointsInRace(int raceId) throws
517         ↪ IDNotRecognisedException {
518         Race race = getRaceById(raceId);
519         List<Rider> riders = race.getRidersByAdjustedElapsedTime();
520         int[] riderIds = new int[riders.size()];
521         for (int i = 0; i < riders.size(); i++) {
522             riderIds[i] = race.getRiderResults(riders.get(i)).getCumulativeMountainPoints();
523         }
524         return riderIds;
525     }
526
527     @Override
528     public int[] getRidersPointClassificationRank(int raceId) throws
529         ↪ IDNotRecognisedException {
530         Race race = getRaceById(raceId);
531         List<Rider> riders = race.getRidersBySprintersPoints();
532         int[] riderIds = new int[riders.size()];
533         for (int i = 0; i < riders.size(); i++) {
534             riderIds[i] = riders.get(i).getId();
535         }
536         return riderIds;
537     }
538
539     @Override
540     public int[] getRidersMountainPointClassificationRank(int raceId)
541         throws IDNotRecognisedException {
542         Race race = getRaceById(raceId);
543         List<Rider> riders = race.getRidersByMountainPoints();
544         int[] riderIds = new int[riders.size()];
545         for (int i = 0; i < riders.size(); i++) {
```

```
542     riderIds[i] = riders.get(i).getId();
543 }
544 return riderIds;
545 }
546 }
```

### 3 IntermediateSprint.java

```
1 package cycling;
2
3 public class IntermediateSprint extends Segment {
4     private final double location;
5
6     public IntermediateSprint(Stage stage, double location)
7         throws InvalidLocationException, InvalidStageTypeException,
8             ↳ InvalidStageStateException {
9         super(stage, SegmentType.SPRINT, location);
10        this.location = location;
11    }
```

### 4 Race.java

```
1 package cycling;
2
3 import java.time.LocalDateTime;
4 import java.util.*;
5 import java.util.stream.Collectors;
6
7 public class Race {
8
9     private final String name;
10    private final String description;
11
12    private final ArrayList<Stage> stages = new ArrayList<>();
13
14    private final HashMap<Rider, RaceResult> results = new HashMap<>();
15
16    private static int count = 0;
17    private final int id;
18
19    public Race(String name, String description) throws InvalidNameException {
20        if (name == null
21            || name.isEmpty()
22            || name.length() > 30
23            || CyclingPortal.containsWhitespace(name)) {
24            throw new InvalidNameException(
25                "The name cannot be null, empty, have more than 30 characters, or have white
26                ↳ spaces.");
27        }
28        this.name = name;
29        this.description = description;
30        this.id = Race.count++;
31    }
32
33    static void resetIdCounter() {
```

```
33     count = 0;
34 }
35
36 static int getIdCounter() {
37     return count;
38 }
39
40 static void setIdCounter(int newCount) {
41     count = newCount;
42 }
43
44 public int getId() {
45     return id;
46 }
47
48 public String getName() {
49     return name;
50 }
51
52 public void addStage(Stage stage) {
53     for (int i = 0; i < stages.size(); i++) {
54         LocalDateTime iStartTime = stages.get(i).getStartTime();
55         if (stage.getStartTime().isBefore(iStartTime)) {
56             stages.add(i, stage);
57             return;
58         }
59     }
60     stages.add(stage);
61 }
62
63 public ArrayList<Stage> getStages() {
64     return stages;
65 }
66
67 public void removeStage(Stage stage) {
68     stages.remove(stage);
69 }
70
71 public String getDetails() {
72     double currentLength = 0;
73     for (final Stage stage : stages) {
74         currentLength = currentLength + stage.getLength();
75     }
76     return ("Race ID: "
77         + id
78         + System.lineSeparator()
79         + "Name: "
80         + name
81         + System.lineSeparator()
82         + "Description: "
83         + description
84         + System.lineSeparator()
85         + "Number of Stages: "
86         + stages.size()
87         + System.lineSeparator()
88         + "Total length: "
89         + currentLength);
90 }
```

```
91
92 public List<Rider> getRidersByAdjustedElapsedTime() {
93     calculateResults();
94     return sortRiderResultsBy(RaceResult.sortByAdjustedElapsedTime);
95 }
96
97 public List<Rider> getRidersBySprintersPoints() {
98     calculateResults();
99     return sortRiderResultsBy(RaceResult.sortBySprintersPoints);
100 }
101
102 public List<Rider> getRidersByMountainPoints() {
103     calculateResults();
104     return sortRiderResultsBy(RaceResult.sortByMountainPoints);
105 }
106
107 public RaceResult getRiderResults(Rider rider) {
108     calculateResults();
109     return results.get(rider);
110 }
111
112 public void removeRiderResults(Rider rider) {
113     results.remove(rider);
114 }
115
116 private List<Rider> sortRiderResultsBy(Comparator<RaceResult> comparison) {
117     return results.entrySet().stream()
118         .sorted(Map.Entry.comparingByValue(comparison))
119         .map(Map.Entry::getKey)
120         .collect(Collectors.toList());
121 }
122
123 private void registerRiderResults(Rider rider, StageResult stageResult) {
124     if (results.containsKey(rider)) {
125         results.get(rider).addStageResult(stageResult);
126     } else {
127         RaceResult raceResult = new RaceResult();
128         raceResult.addStageResult(stageResult);
129         results.put(rider, raceResult);
130     }
131 }
132
133 private void calculateResults() {
134     for (Stage stage : stages) {
135         HashMap<Rider, StageResult> stageResults = stage.getStageResults();
136         for (Rider rider : stageResults.keySet()) {
137             registerRiderResults(rider, stageResults.get(rider));
138         }
139     }
140 }
141 }
```

## 5 RaceResult.java

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

```

4  import java.time.LocalTime;
5  import java.util.Comparator;
6
7  public class RaceResult {
8      private Duration cumulativeAdjustedElapsedTime = Duration.ZERO;
9      private int cumulativeSprintersPoints = 0;
10     private int cumulativeMountainPoints = 0;
11
12     // TODO: Test ordered Asc
13     protected static final Comparator<RaceResult> sortByAdjustedElapsedTime =
14         Comparator.comparing(RaceResult::getCumulativeAdjustedElapsedTime);
15
16     // TODO: Test order Desc
17     protected static final Comparator<RaceResult> sortBySprintersPoints =
18         Comparator.comparing(RaceResult::getCumulativeSprintersPoints).reversed();
19     // protected static final Comparator<RaceResult> sortBySprintersPoints = (RaceResult
20     ↪ result1,
21     //     RaceResult result2) ->
22     ↪ Integer.compare(result2.getCumulativeSprintersPoints(),
23     //     result1.getCumulativeSprintersPoints());
24     protected static final Comparator<RaceResult> sortByMountainPoints =
25         Comparator.comparing(RaceResult::getCumulativeMountainPoints).reversed();
26     // protected static final Comparator<RaceResult> sortByMountainPoints = (RaceResult
27     ↪ result1,
28     //     RaceResult result2) -> Integer.compare(result2.getCumulativeMountainPoints(),
29     //     result1.getCumulativeMountainPoints());
30
31     public Duration getCumulativeAdjustedElapsedTime() {
32         return this.cumulativeAdjustedElapsedTime;
33     }
34
35     public LocalTime getCumulativeAdjustedElapsedLocalTime() {
36         return LocalTime.MIDNIGHT.plus(this.cumulativeAdjustedElapsedTime);
37     }
38
39     public int getCumulativeMountainPoints() {
40         return this.cumulativeMountainPoints;
41     }
42
43     public int getCumulativeSprintersPoints() {
44         return this.cumulativeSprintersPoints;
45     }
46
47     public void addStageResult(StageResult stageResult) {
48         this.cumulativeAdjustedElapsedTime =
49             this.cumulativeAdjustedElapsedTime.plus(stageResult.getAdjustedElapsedTime());
50         this.cumulativeSprintersPoints += stageResult.getSprintersPoints();
51         this.cumulativeMountainPoints += stageResult.getMountainPoints();
52     }
53 }

```

## 6 Rider.java

```

1  package cycling;
2
3  public class Rider {
4      private final Team team;

```

```
5     private final String name;
6     private final int yearOfBirth;
7
8     private static int count = 0;
9     private final int id;
10
11    public Rider(Team team, String name, int yearOfBirth) throws
12        ↪ IllegalArgumentException {
13        if (name == null) {
14            throw new java.lang.IllegalArgumentException("The rider's name cannot be
15                ↪ null.");
16        }
17        if (yearOfBirth < 1900) {
18            throw new java.lang.IllegalArgumentException(
19                "The rider's birth year is invalid, must be greater than 1900.");
20        }
21
22        this.team = team;
23        this.name = name;
24        this.yearOfBirth = yearOfBirth;
25        this.id = Rider.count++;
26    }
27
28    static void resetIdCounter() {
29        count = 0;
30    }
31
32    static int getIdCounter() {
33        return count;
34    }
35
36    static void setIdCounter(int newCount) {
37        count = newCount;
38    }
39
40    public int getId() {
41        return id;
42    }
43
44    public Team getTeam() {
45        return team;
46    }
47 }
```

## 7 Segment.java

```
1 package cycling;
2
3 import java.time.LocalDateTime;
4 import java.util.HashMap;
5 import java.util.List;
6 import java.util.Map;
7 import java.util.stream.Collectors;
8
9 public class Segment {
10     private static int count = 0;
11     private final Stage stage;
```



```
12     private final int id;
13     private final SegmentType type;
14     private final double location;
15
16     private final HashMap<Rider, SegmentResult> results = new HashMap<>();
17
18     private static final int[] SPRINT_POINTS = {20, 17, 15, 13, 11, 10, 9, 8, 7, 6, 5,
19     ↪ 4, 3, 2, 1};
20     private static final int[] HC_POINTS = {20, 15, 12, 10, 8, 6, 4, 2};
21     private static final int[] C1_POINTS = {10, 8, 6, 4, 2, 1};
22     private static final int[] C2_POINTS = {5, 3, 2, 1};
23     private static final int[] C3_POINTS = {2, 1};
24     private static final int[] C4_POINTS = {1};
25
26     public Segment(Stage stage, SegmentType type, double location)
27         throws InvalidLocationException, InvalidStageStateException,
28         ↪ InvalidStageTypeException {
29         if (location > stage.getLength()) {
30             throw new InvalidLocationException("The location is out of bounds of the stage
31             ↪ length.");
32         }
33         if (stage.isWaitingForResults()) {
34             throw new InvalidStageStateException("The stage is waiting for results.");
35         }
36         if (stage.getType().equals(StageType.TT)) {
37             throw new InvalidStageTypeException("Time-trial stages cannot contain any
38             ↪ segments.");
39         }
40         this.stage = stage;
41         this.id = Segment.count++;
42         this.type = type;
43         this.location = location;
44     }
45
46     static void resetIdCounter() {
47         count = 0;
48     }
49
50     static int getIdCounter() {
51         return count;
52     }
53
54     static void setIdCounter(int newCount) {
55         count = newCount;
56     }
57
58     public SegmentType getType() {
59         return type;
60     }
61
62     public int getId() {
63         return id;
64     }
65
66     public Stage getStage() {
67         return stage;
68     }
```

```
66     public double getLocation() {
67         return location;
68     }
69
70     public void registerResults(Rider rider, LocalTime finishTime) {
71         SegmentResult result = new SegmentResult(finishTime);
72         results.put(rider, result);
73     }
74
75     public SegmentResult getRiderResult(Rider rider) {
76         calculateResults();
77         return results.get(rider);
78     }
79
80     public void removeRiderResults(Rider rider) {
81         results.remove(rider);
82     }
83
84     private List<Rider> sortRiderResults() {
85         return results.entrySet().stream()
86             .sorted(Map.Entry.comparingByValue(SegmentResult.sortByFinishTime))
87             .map(Map.Entry::getKey)
88             .collect(Collectors.toList());
89     }
90
91     private void calculateResults() {
92         List<Rider> riders = sortRiderResults();
93
94         for (int i = 0; i < results.size(); i++) {
95             Rider rider = riders.get(i);
96             SegmentResult result = results.get(rider);
97             int position = i + 1;
98             // Position Calculation
99             result.setPosition(position);
100
101             // Points Calculation
102             int[] pointsDistribution = getPointsDistribution();
103             if (position <= pointsDistribution.length) {
104                 int points = pointsDistribution[i];
105                 if (this.type.equals(SegmentType.SPRINT)) {
106                     result.setSprintersPoints(points);
107                     result.setMountainPoints(0);
108                 } else {
109                     result.setSprintersPoints(0);
110                     result.setMountainPoints(points);
111                 }
112             } else {
113                 result.setMountainPoints(0);
114                 result.setSprintersPoints(0);
115             }
116         }
117     }
118
119     private int[] getPointsDistribution() {
120         return switch (type) {
121             case HC -> HC_POINTS;
122             case C1 -> C1_POINTS;
123             case C2 -> C2_POINTS;
```

```
124         case C3 -> C3_POINTS;
125         case C4 -> C4_POINTS;
126         case SPRINT -> SPRINT_POINTS;
127     };
128 }
129 }
```

## 8 SegmentResult.java

```
1  package cycling;
2
3  import java.time.LocalDateTime;
4  import java.util.Comparator;
5
6  public class SegmentResult {
7      private final LocalDateTime finishTime;
8      private int position;
9      private int sprintersPoints;
10     private int mountainPoints;
11
12     protected static final Comparator<SegmentResult> sortByFinishTime =
13         Comparator.comparing(SegmentResult::getFinishTime);
14
15     public SegmentResult(LocalDateTime finishTime) {
16         this.finishTime = finishTime;
17     }
18
19     public LocalDateTime getFinishTime() {
20         return finishTime;
21     }
22
23     public void setPosition(int position) {
24         this.position = position;
25     }
26
27     public int getPosition() {
28         return position;
29     }
30
31     public void setMountainPoints(int points) {
32         this.mountainPoints = points;
33     }
34
35     public void setSprintersPoints(int points) {
36         this.sprintersPoints = points;
37     }
38
39     public int getMountainPoints() {
40         return this.mountainPoints;
41     }
42
43     public int getSprintersPoints() {
44         return this.sprintersPoints;
45     }
46 }
```

## 9 Stage.java

```
1  package cycling;
2
3  import java.time.Duration;
4  import java.time.LocalDateTime;
5  import java.time.LocalTime;
6  import java.util.ArrayList;
7  import java.util.HashMap;
8  import java.util.List;
9  import java.util.Map;
10 import java.util.stream.Collectors;
11
12 public class Stage {
13     private final Race race;
14     private final String name;
15     private final String description;
16     private final double length;
17     private final LocalDateTime startTime;
18     private final StageType type;
19     private final int id;
20     private static int count = 0;
21     private boolean waitingForResults = false;
22     private final ArrayList<Segment> segments = new ArrayList<>();
23
24     private final HashMap<Rider, StageResult> results = new HashMap<>();
25
26     private static final int[] FLAT_POINTS = {50, 30, 20, 18, 16, 14, 12, 10, 8, 7, 6,
27     ↪ 5, 4, 3, 2};
28     private static final int[] MEDIUM_POINTS = {30, 25, 22, 19, 17, 15, 13, 11, 9, 7, 6,
29     ↪ 5, 4, 3, 2};
30     private static final int[] HIGH_POINTS = {20, 17, 15, 13, 11, 10, 9, 8, 7, 6, 5, 4,
31     ↪ 3, 2, 1};
32     private static final int[] TT_POINTS = {20, 17, 15, 13, 11, 10, 9, 8, 7, 6, 5, 4, 3,
33     ↪ 2, 1};
34
35     public Stage(
36         Race race,
37         String name,
38         String description,
39         double length,
40         LocalDateTime startTime,
41         StageType type)
42         throws InvalidNameException, InvalidLengthException {
43         if (name == null
44             || name.isEmpty()
45             || name.length() > 30
46             || CyclingPortal.containsWhitespace(name)) {
47             throw new InvalidNameException(
48                 "Stage name cannot be null, empty, have more than 30 characters or have
49                 ↪ white spaces.");
50         }
51         if (length < 5) {
52             throw new InvalidLengthException("Length is invalid, cannot be less than 5km.");
53         }
54         this.name = name;
55         this.description = description;
56         this.race = race;
```

```
52     this.length = length;
53     this.startTime = startTime;
54     this.type = type;
55     this.id = Stage.count++;
56 }
57
58 static void resetIdCounter() {
59     count = 0;
60 }
61
62 static int getIdCounter() {
63     return count;
64 }
65
66 static void setIdCounter(int newCount) {
67     count = newCount;
68 }
69
70 public int getId() {
71     return id;
72 }
73
74 public String getName() {
75     return name;
76 }
77
78 public double getLength() {
79     return length;
80 }
81
82 public Race getRace() {
83     return race;
84 }
85
86 public StageType getType() {
87     return type;
88 }
89
90 public ArrayList<Segment> getSegments() {
91     return segments;
92 }
93
94 public LocalDateTime getStartTime() {
95     return startTime;
96 }
97
98 public void addSegment(Segment segment) {
99     for (int i = 0; i < segments.size(); i++) {
100         if (segment.getLocation() < segments.get(i).getLocation()) {
101             segments.add(i, segment);
102             return;
103         }
104     }
105     segments.add(segment);
106 }
107
108 public void removeSegment(Segment segment) throws InvalidStageStateException {
109     if (waitingForResults) {
```

```
110         throw new InvalidStageStateException(  
111             "The stage cannot be removed as it is waiting for results.");  
112     }  
113     segments.remove(segment);  
114 }  
115  
116 public void registerResult(Rider rider, LocalTime[] checkpoints)  
117     throws InvalidStageStateException, DuplicatedResultException,  
118         InvalidCheckpointsException {  
119     if (!waitingForResults) {  
120         throw new InvalidStageStateException(  
121             "Results can only be added to a stage while it is waiting for results.");  
122     }  
123     if (results.containsKey(rider)) {  
124         throw new DuplicatedResultException("Each rider can only have one result per  
125             ↪ Stage.");  
126     }  
127     if (checkpoints.length != segments.size() + 2) {  
128         throw new InvalidCheckpointsException(  
129             "The length of the checkpoint must equal number of Segments in the Stage +  
130             ↪ 2.");  
131     }  
132  
133     StageResult result = new StageResult(checkpoints);  
134     // Save Riders result for the Stage  
135     results.put(rider, result);  
136  
137     // Propagate all the Riders results for each segment  
138     for (int i = 0; i < segments.size(); i++) {  
139         segments.get(i).registerResults(rider, checkpoints[i + 1]);  
140     }  
141 }  
142  
143 public void concludePreparation() throws InvalidStageStateException {  
144     if (waitingForResults) {  
145         throw new InvalidStageStateException("Stage is already waiting for results.");  
146     }  
147     waitingForResults = true;  
148 }  
149  
150 public boolean isWaitingForResults() {  
151     return waitingForResults;  
152 }  
153  
154 public StageResult getRiderResult(Rider rider) {  
155     calculateResults();  
156     return results.get(rider);  
157 }  
158  
159 public void removeRiderResults(Rider rider) {  
160     results.remove(rider);  
161 }  
162  
163 public List<Rider> getRidersByElapsedTime() {  
164     calculateResults();  
165     return sortRiderResults();  
166 }
```

```
165     public HashMap<Rider, StageResult> getStageResults() {
166         calculateResults();
167         return results;
168     }
169
170     private List<Rider> sortRiderResults() {
171         return results.entrySet().stream()
172             .sorted(Map.Entry.comparingByValue(StageResult.sortByElapsedTime))
173             .map(Map.Entry::getKey)
174             .collect(Collectors.toList());
175     }
176
177     private void calculateResults() {
178         List<Rider> riders = sortRiderResults();
179
180         for (int i = 0; i < results.size(); i++) {
181             Rider rider = riders.get(i);
182             StageResult result = results.get(rider);
183             int position = i + 1;
184
185             // Position Calculation
186             result.setPosition(position);
187
188             // Adjusted Elapsed Time Calculations
189             if (i == 0) {
190                 result.setAdjustedElapsedTime(result.getElapsedTime());
191             } else {
192                 Rider prevRider = riders.get(i - 1);
193                 Duration prevTime = results.get(prevRider).getElapsedTime();
194                 Duration time = results.get(rider).getElapsedTime();
195
196                 int timeDiff = time.minus(prevTime).compareTo(Duration.ofSeconds(1));
197                 if (timeDiff <= 0) {
198                     // Close Finish Condition
199                     Duration prevAdjustedTime = results.get(prevRider).getAdjustedElapsedTime();
200                     result.setAdjustedElapsedTime(prevAdjustedTime);
201                 } else {
202                     // Far Finish Condition
203                     result.setAdjustedElapsedTime(time);
204                 }
205             }
206
207             // Points Calculation
208             int sprintersPoints = 0;
209             int mountainPoints = 0;
210             for (Segment segment : segments) {
211                 SegmentResult segmentResult = segment.getRiderResult(rider);
212                 sprintersPoints += segmentResult.getSprintersPoints();
213                 mountainPoints += segmentResult.getMountainPoints();
214             }
215             int[] pointsDistribution = getPointDistribution();
216             if (position <= pointsDistribution.length) {
217                 sprintersPoints += pointsDistribution[i];
218             }
219             result.setSprintersPoints(sprintersPoints);
220             result.setMountainPoints(mountainPoints);
221         }
222     }
```

```

223
224     private int[] getPointDistribution() {
225         return switch (type) {
226             case FLAT -> FLAT_POINTS;
227             case MEDIUM_MOUNTAIN -> MEDIUM_POINTS;
228             case HIGH_MOUNTAIN -> HIGH_POINTS;
229             case TT -> TT_POINTS;
230         };
231     }
232 }

```

## 10 StageResult.java

```

1  package cycling;
2
3  import java.time.Duration;
4  import java.time.LocalDateTime;
5  import java.util.Comparator;
6
7  public class StageResult {
8      private final LocalDateTime[] checkpoints;
9      private final Duration elapsedTime;
10     private Duration adjustedElapsedTime;
11     private int position;
12     private int sprintersPoints;
13     private int mountainPoints;
14
15     protected static final Comparator<StageResult> sortByElapsedTime =
16         Comparator.comparing(StageResult::getElapsedTime);
17
18     public StageResult(LocalDateTime[] checkpoints) {
19         this.checkpoints = checkpoints;
20         this.elapsedTime = Duration.between(checkpoints[0], checkpoints[checkpoints.length
21         ↪ - 1]);
22     }
23
24     public LocalDateTime[] getCheckpoints() {
25         return this.checkpoints;
26     }
27
28     public Duration getElapsedTime() {
29         return elapsedTime;
30     }
31
32     public void setPosition(int position) {
33         this.position = position;
34     }
35
36     public void setAdjustedElapsedTime(Duration adjustedElapsedTime) {
37         this.adjustedElapsedTime = adjustedElapsedTime;
38     }
39
40     public int getPosition() {
41         return position;
42     }
43
44     public Duration getAdjustedElapsedTime() {

```



```

44     return adjustedElapsedTime;
45 }
46
47 public LocalTime getAdjustedElapsedLocalTime() {
48     return checkpoints[0].plus(adjustedElapsedTime);
49 }
50
51 public void setMountainPoints(int points) {
52     this.mountainPoints = points;
53 }
54
55 public void setSprintersPoints(int points) {
56     this.sprintersPoints = points;
57 }
58
59 public int getMountainPoints() {
60     return mountainPoints;
61 }
62
63 public int getSprintersPoints() {
64     return sprintersPoints;
65 }
66
67 // --Commented out by Inspection START (28/03/2022, 3:31 pm):
68 // public void add(StageResult res){
69 //     this.elapsedTime = this.elapsedTime.plus(res.getElapsedTime());
70 //     this.adjustedElapsedTime =
71 // ↪ this.adjustedElapsedTime.plus(res.getAdjustedElapsedTime());
72 //     this.sprintersPoints += res.getSprintersPoints();
73 //     this.mountainPoints += res.getMountainPoints();
74 // }
75 // --Commented out by Inspection STOP (28/03/2022, 3:31 pm)

```

## 11 Team.java

```

1 package cycling;
2
3 import java.util.ArrayList;
4
5 public class Team {
6     private final String name;
7     private final String description;
8
9     private final ArrayList<Rider> riders = new ArrayList<>();
10    private static int count = 0;
11    private final int id;
12
13    public Team(String name, String description) throws InvalidNameException {
14        if (name == null
15            || name.isEmpty()
16            || name.length() > 30
17            || CyclingPortal.containsWhitespace(name)) {
18            throw new InvalidNameException(
19                "Team name cannot be null, empty, have more than 30 characters or have white
20                ↪ spaces.");
21        }
22    }

```

```
21     this.name = name;
22     this.description = description;
23     this.id = Team.count++;
24 }
25
26 static void resetIdCounter() {
27     count = 0;
28 }
29
30 static int getIdCounter() {
31     return count;
32 }
33
34 static void setIdCounter(int newCount) {
35     count = newCount;
36 }
37
38 public String getName() {
39     return name;
40 }
41
42 public int getId() {
43     return id;
44 }
45
46 public void removeRider(Rider rider) {
47     riders.remove(rider);
48 }
49
50 public ArrayList<Rider> getRiders() {
51     return riders;
52 }
53
54 public void addRider(Rider rider) {
55     riders.add(rider);
56 }
57 }
```