1 Team.java

```
1  package cycling;
2  import java.util.ArrayList;
3
4  public class Team {
5      //instance attributes
6      private String teamName;
7      private String teamDesc;
8      private int teamID;
9
10
11
12     /**
13      * method: get team ID
14      * @return int
15      */
16     public int getTeamID() {
17         return this.teamID;
18     }
19
20
21     /**
22      * method: get team name
23      * @return String
24      */
25     public String getTeamName() {
26         return this.teamName;
27     }
28
29
30     /**
31      * method: get team desc
32      * @return String
33      */
34     public String getTeamDesc() {
35         return this.teamDesc;
36     }
37
38     //contructors
39     public Team() {
40         this.teamName = "Null";
41         this.teamDesc = "Null";
42         this.teamID = 0;
43     }
44
45     public Team(String teamName, String teamDesc, ArrayList<Team> teamList) {
46         this.teamName = teamName;
47         this.teamDesc = teamDesc;
48
49         if (teamList.isEmpty()){
50             this.teamID = 2000;
51         }
52         else{
```

```
            //set ID 1 more than last team
            this.teamID = teamList.get(teamList.size() - 1).getTeamID() + 1;
        }
    }
}
```

2 Rider.java

```
package cycling;

import java.util.ArrayList;

public class Rider implements java.io.Serializable{

    //instance attributes
    private int riderID;
    private int teamID;
    private String riderName;
    private int yearOfBirth;


    /**
     * method: get rider ID
     * @return int
     */
    public int getRiderID() {
        return riderID;
    }


    /**
     * method: get team ID
     * @return int
     */
    public int getTeamID() {
        return teamID;
    }


    /**
     * method: get rider name
     * @return String
     */
    public String getRiderName() {
        return riderName;
    }


    /**
     * method: get year of birth
     * @return int
     */
    public int getYearOfBirth() {
        return yearOfBirth;
```

```
    }

    //contructor
    public Rider() {
        teamID = 0;
        riderName = "Null";
        yearOfBirth = 0;
        riderID = 0;
    }
    public Rider(int teamID, int yearOfBirth, String riderName, ArrayList<Rider> riderList) {

        //set instance attributes
        this.teamID = teamID;
        this.riderName = riderName;
        this.yearOfBirth = yearOfBirth;

        if (riderList.isEmpty()){
            this.riderID = 2000;
        }
        else{
            //set ID 1 more than last rider
            this.riderID = riderList.get(riderList.size() - 1).getRiderID() + 1;
        }
    }
}
```

3 Race.java

```
package cycling;
import java.util.ArrayList;

public class Race {

    //instance attributes
    private int raceID;
    private String raceName;
    private String raceDesc;
    private int numOfStages = 0;
    private double totalLength = 0;


    /**
     * toString
     * @return String
     */
    public String toString() {
        return ("Race ID: " + raceID + "\nName: " + raceName + "\nDescription: " + raceDesc +
        "\nNumber of Stages: " + numOfStages + "\nTotal Length: " + totalLength);
    }


    /**
     * get race ID
     * @return int
```

```
27          */
28         public int getRaceID(){
29             return this.raceID;
30         }
31 
32 
33         /**
34          * get race name
35          * @return String
36          */
37         public String getRaceName(){
38             return this.raceName;
39         }
40 
41 
42         /**
43          * get race description
44          * @return String
45          */
46         public String getRaceDesc(){
47             return this.raceDesc;
48         }
49 
50 
51         /**
52          * get total length
53          * @return double
54          */
55         public double getTotalLength() {
56             return this.totalLength;
57         }
58 
59 
60         /**
61          * get number of stages
62          * @return int
63          */
64         public int getNumOfStages() {
65             return this.numOfStages;
66         }
67 
68 
69         /** ,
70          * set number of stages
71          * @param stageList
72          */
73         public void setNumOfStages(ArrayList<Stage> stageList) {
74             numOfStages = 0;
75             for (int i = 0; i < stageList.size(); i++) {
76                 if (stageList.get(i).getRaceID() == this.raceID) {
77                     this.numOfStages ++;
78                 }
79             }
80         }
81 
```

```
82
83      /**
84       * set total length
85       * @param stageList
86       */
87      public void setTotalLength(ArrayList<Stage> stageList) {
88          totalLength = 0;
89          for (int i = 0; i < stageList.size(); i++) {
90              if (stageList.get(i).getRaceID() == this.raceID) {
91                  this.totalLength += stageList.get(i).getStageLength();
92              }
93          }
94      }
95
96
97      //constructor
98      public Race(String raceName, String raceDesc, ArrayList<Race> raceList) {
99          this.raceName = raceName;
100         this.raceDesc = raceDesc;
101
102         if (raceList.isEmpty()){
103             this.raceID = 2000;
104         }
105         else{
106             //set ID 1 more than last race
107             this.raceID = raceList.get(raceList.size() - 1).getRaceID() + 1;
108         }
109     }
110 }
```

4 Stage.java

```
1  package cycling;
2  import java.sql.Time;
3  import java.util.ArrayList;
4  import java.time.LocalDateTime;
5
6  public class Stage {
7
8      //instance attributes
9      private int stageID;
10     private String stageName;
11     private String stageDesc;
12     private double stageLength;
13     private LocalDateTime startTime;
14     private StageType stageType;
15     private int raceID;
16     private String stageState;
17
18
19     /**
20      * method: get stageID
21      * @return int
22      */
```

```
23      public int getStageID() {
24          return this.stageID;
25      }
26
27
28      /**
29       * method: get race ID
30       * @return int
31       */
32      public int getRaceID(){
33          return this.raceID;
34      }
35
36
37      /**
38       * method: get stageName
39       * @return String
40       */
41      public String getStageName() {
42          return this.stageName;
43      }
44
45
46      /**
47       * method: get stageLength
48       * @return double
49       */
50      public double getStageLength() {
51          return this.stageLength;
52      }
53
54
55      /**
56       * method: get stageType
57       * @return StageType
58       */
59      public StageType getStageType() {
60          return this.stageType;
61      }
62
63
64      /**
65       * method: get stageState
66       * @return String
67       */
68      public String getStageState() {
69          return this.stageState;
70      }
71
72
73      /**
74       * method: set stageState
75       * @param state
76       */
77      public void setStageState(String state) {
```

```
78          this.stageState = state;
79      }
80
81      //contructors
82      public Stage() {
83
84      }
85
86      public Stage(int raceID, String stageName, String stageDesc, double stageLength, LocalDateTime
            startTime, StageType stageType, ArrayList<Stage> stageList) {
87          this.stageName = stageName;
88          this.stageDesc = stageDesc;
89          this.stageLength = stageLength;
90          this.startTime = startTime;
91          this.stageType = stageType;
92          this.raceID= raceID;
93
94          if (stageList.isEmpty()){
95              this.stageID = 2000;
96          }
97          else{
98              //set ID 1 more than last stage
99              this.stageID = stageList.get(stageList.size() - 1).getStageID() + 1;
100         }
101     }
102 }
```

5 Segment.java

```
1  package cycling;
2  import java.util.ArrayList;
3
4  public class Segment {
5
6      //instance attrbutes
7      private int segmentID;
8      private int stageID;
9      private double location;
10     private SegmentType type;
11     private double averageGradient;
12     private double length;
13
14
15     /**
16      * method: get segment id
17      * @return int
18      */
19     public int getSegmentID() {
20         return this.segmentID;
21     }
22
23
24     /**
25      * method: get stage id
```

```
26          * @return int
27          */
28         public int getStageID() {
29             return this.stageID;
30         }
31 
32 
33         /**
34          * method: get segment type
35          * @return SegmentType
36          */
37         public SegmentType getSegmentType() {
38             return this.type;
39         }
40 
41 
42         /**
43          * method: get segment location
44          * @return double
45          */
46         public double getSegmentLocation() {
47             return this.location;
48         }
49 
50         //constructor
51         public Segment(int stageID, double location, ArrayList<Segment> segmentList) {
52             this.stageID = stageID;
53             this.location = location;
54             this.type = SegmentType.SPRINT;
55 
56             if (segmentList.isEmpty()){
57                 this.segmentID = 2000;
58             }
59             else{
60                 //set ID 1 more than last segment
61                 this.segmentID = segmentList.get(segmentList.size() - 1).getSegmentID() + 1;
62             }
63         }
64 
65         public Segment(int stageID, double location, SegmentType type, double averageGradient, double length,
               ArrayList<Segment> segmentList ) {
66 
67             this.stageID = stageID;
68             this.location = location;
69             this.type = type;
70             this.averageGradient = averageGradient;
71             this.length = length;
72 
73             if (segmentList.isEmpty()){
74                 this.segmentID = 2000;
75             }
76             else{
77                 //set ID 1 more than last segment
78                 this.segmentID = segmentList.get(segmentList.size() - 1).getSegmentID() + 1;
79             }
```

```
    }
}
```

6 Result.java

```
package cycling;

import java.time.LocalTime;
import java.util.ArrayList;

public class Result {

    //instance attributes
    private int resultID;
    private int stageID;
    private int riderID;
    private LocalTime[] checkpoints;


    /**
     * method: get resultID
     * @return int
     */
    public int getResultID() {
        return this.resultID;
    }


    /**
     * method: get riderID
     * @return int
     */
    public int getRiderID() {
        return this.riderID;
    }


    /**
     * method: get stageID
     * @return int
     */
    public int getStageID() {
        return this.stageID;
    }


    /**
     * method: get checkpoints
     * @return LocalTime[]
     */
    public LocalTime[] getCheckpoints() {
        return this.checkpoints;
    }

```

```
    //constructor
    public Result(int stageID, int riderID, LocalTime[] checkpoints, ArrayList<Result> resultList) {
        this.stageID = stageID;
        this.riderID = riderID;
        this.checkpoints = checkpoints;

        if (resultList.isEmpty()){
            this.resultID = 2000;
        }
        else{
            //set ID 1 more than last team
            this.resultID = resultList.get(resultList.size() - 1).getResultID() + 1;
        }

    }




}
```

7 CyclingPortal.java

```
package cycling;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.time.Duration;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.time.chrono.*;




/**
 * CyclingPortal is an implementor
 * of the CyclingPortalInterface interface.
 *
 * @author 710030198
 * @author 710045992
 * @version 1.0
 *
 */
public class CyclingPortal implements CyclingPortalInterface {

   //object arrays
   ArrayList<Rider> riderList = new ArrayList<>();
```

```
31      ArrayList<Stage> stageList = new ArrayList<>();
32      ArrayList<Segment> segmentList = new ArrayList<>();
33      ArrayList<Race> raceList = new ArrayList<>();
34      ArrayList<Team> teamList = new ArrayList<>();
35      ArrayList<Result> resultList = new ArrayList<>();
36
37      /**
38       * Checks the name of a race exists
39       * @param name
40       * @return validName
41       */
42      public boolean checkRaceName(String name) {
43         boolean validName = false;
44
45         //for every race
46         for (int i = 0; i < raceList.size(); i++) {
47            //if race exists
48            if (raceList.get(i).getRaceName() == name) {
49               validName = true;
50            }
51         }
52
53         return validName;
54      }
55
56      /**
57       * Checks the id of a race exists
58       * @param raceID
59       * @return validID
60       */
61      public boolean checkRaceID(int raceID) {
62
63         boolean validID = false;
64
65         //for every race
66         for (int i = 0; i < raceList.size(); i++) {
67            //if race exists
68            if (raceList.get(i).getRaceID() == raceID) {
69               //race is valid
70               validID = true;
71            }
72         }
73
74         return validID;
75      }
76
77      /**
78       * Checks the id of a rider exists
79       * @param riderID
80       * @return validID
81       */
82      public boolean checkRiderID(int riderID) {
83
84         boolean validID = false;
85
```

```
86          //for every rider
87          for (int i = 0; i < riderList.size(); i++) {
88             //if rider exists
89             if (riderList.get(i).getRiderID() == riderID) {
90                //rider is valid
91                validID = true;
92             }
93          }
94
95          return validID;
96       }
97
98       /**
99        * Checks the id of a team exists
100       * @param teamID
101       * @return validID
102       */
103      public boolean checkTeamID(int teamID) {
104
105         boolean validID = false;
106
107         //for every team
108         for (int i = 0; i < teamList.size(); i++) {
109            //if team exists
110            if (teamList.get(i).getTeamID() == teamID) {
111               //team is valid
112               validID = true;
113            }
114         }
115
116         return validID;
117      }
118
119      /**
120       * Checks the id of a stage exists
121       * @param stageID
122       * @return validID
123       */
124      public boolean checkStageID(int stageID) {
125
126         boolean validID = false;
127
128         //for every stage
129         for (int i = 0; i < stageList.size(); i++) {
130            //if stage exists
131            if (stageList.get(i).getStageID() == stageID) {
132               //stage is valid
133               validID = true;
134            }
135         }
136
137         return validID;
138      }
139
140      /**
```

```
141      * Checks the id of a segment exists
142      * @param segmentID
143      * @return validID
144      */
145     public boolean checkSegmentID(int segmentID) {
146
147        boolean validID = false;
148
149        //for every segement
150        for (int i = 0; i < segmentList.size(); i++) {
151           //if segment exists
152           if (segmentList.get(i).getSegmentID() == segmentID) {
153              //segment is valid
154              validID = true;
155           }
156        }
157
158        return validID;
159     }
160
161     /**
162      * Checks if stage is a time trial
163      * @param stageID
164      * @return validStage
165      */
166     public boolean checkStageType(int stageID) {
167
168        boolean validStage = true;
169
170        //for every stage
171        for (int i = 0; i < stageList.size(); i++) {
172           //if stage exists
173           if (stageList.get(i).getStageID() == stageID) {
174              //if stage is time trials
175              if ( stageList.get(i).getStageType() == StageType.TT) {
176                 //throw exception
177                 validStage = false;
178              }
179           }
180        }
181
182        return validStage;
183     }
184
185
186     /**
187      * Checks the location of segment
188      * @param stageId
189      * @param location
190      * @return validLocation
191      */
192     public boolean checkSegmentLocation(int stageID, double location) {
193        boolean validLocation = true;
194
195        //for every segment
```

```
196         for (int i = 0; i < stageList.size(); i++) {
197            //if stage exists
198            if (stageList.get(i).getStageID() == stageID) {
199               //if invalid length
200               if (stageList.get(i).getStageLength() < location) {
201                  validLocation = false;
202               }
203            }
204         }
205
206         return validLocation;
207      }
208
209      /**
210       * Checks the state of stage
211       * @param stageID
212       * @return validState
213       */
214      public boolean checkStageState(int stageID) {
215         boolean validState = true;
216
217         //for every stage
218         for (int i = 0; i < stageList.size(); i++) {
219            //if stage exists
220            if (stageList.get(i).getStageID() == stageID) {
221               //if stage is waiting for results
222               if (stageList.get(i).getStageState() == "waiting for results") {
223                  validState = false;
224               }
225            }
226         }
227
228         return validState;
229      }
230
231      /**
232       * Gets the total elapsed time
233       * @param riderID
234       * @param stageID
235       * @return totalTime
236       */
237       public LocalTime getTotalTime(int riderID, int stageID) {
238
239         long tempTime = 0;
240
241         LocalTime totalTime = LocalTime.of(0, 0, 0);
242         //for every result
243         for (int i = 0; i < resultList.size(); i++) {
244            //if result is part of rider and stage
245            if ((resultList.get(i).getRiderID() == riderID) &&
246            (resultList.get(i).getStageID() == stageID)) {
247               //get start and finish time
248               LocalTime[] checkPoints = resultList.get(i).getCheckpoints();
249               LocalTime startTime = checkPoints[0];
250               LocalTime finishTime = checkPoints[checkPoints.length - 1];
```

```
251
252             //get difference between start and finish time
253             tempTime = Duration.between(startTime, finishTime).toSeconds();
254
255             //convert to (int) seconds and mins
256             int seconds = (int)tempTime % 60;
257             int mins = (int)tempTime / 60;
258
259             //convert to LocalTime
260             totalTime = LocalTime.of(0, mins, seconds);
261          }
262       }
263       return totalTime;
264    }
265
266
267    /**
268     * Gets an array of all race id's
269     * @return raceIDlist
270     */
271    @Override
272    public int[] getRaceIds() {
273
274       //create array of correct size
275       int [] raceIDList = new int[raceList.size()];
276       if (raceList.size() > 0 ) {
277          for(int i = 0;i < raceList.size(); i++) {
278             //fill array with race IDs
279             raceIDList[i]= raceList.get(i).getRaceID();
280          }
281          //check first ID is correct
282          assert raceIDList[0] == raceList.get(0).getRaceID();
283       }
284
285       return raceIDList;
286    }
287
288    /**
289     * Creates a new race object
290     * @param name
291     * @param description
292     * @return validID
293     */
294    @Override
295    public int createRace(String name, String description) throws IllegalNameException, InvalidNameException
           {
296
297       //for every race
298       for(int i = 0;i < raceList.size(); i++) {
299          //if race exists
300          if (raceList.get(i).getRaceName() == name) {
301             //throw exception
302             throw new IllegalNameException();
303          }
304       }
```

```
305
306         //if name has invalid attributes
307         if((name == null) || (name == "") || (name.length() > 30) || (name.contains(" "))) {
308             //throw exception
309             throw new InvalidNameException();
310         }
311
312         //add new race to raceList
313         raceList.add(new Race(name, description, raceList));
314         //assert race was added
315         assert raceList.get(raceList.size() - 1).getRaceName() == name;
316         //return race ID
317         return raceList.get(raceList.size() - 1).getRaceID();
318
319
320     }
321
322     /**
323      * Gets details from a race
324      * @param raceId
325      * @return Formatted ArrayList Object raceList
326      */
327     @Override
328     public String viewRaceDetails(int raceId) throws IDNotRecognisedException {
329
330         //if race doesn't exist
331         if (checkRaceID(raceId) == false) {
332             //throw exception
333             throw new IDNotRecognisedException();
334         }
335
336         int correctRace = 0;
337         //for all races
338         for (int i = 0; i < raceList.size(); i++) {
339             //if race exists
340             if (raceList.get(i).getRaceID() == raceId) {
341                 //set correctRace
342                 correctRace = i;
343                 break;
344             }
345         }
346
347         //return formatted race details
348         return raceList.get(correctRace).toString();
349
350
351     }
352
353     /**
354      * Removes a race
355      * @param raceId
356      * @return validID
357      */
358     @Override
359     public void removeRaceById(int raceId) throws IDNotRecognisedException {
```

```
360
361         //if race doesn't exist
362         if (checkRaceID(raceId) == false) {
363             //throw exception
364             throw new IDNotRecognisedException();
365         }
366
367         //for every race
368         for (int i = 0; i < raceList.size(); i++) {
369             //if race exists
370             if (raceList.get(i).getRaceID() == raceId) {
371                 //remove stages
372                 //for every stage
373                 for (int j = 0; j < stageList.size(); j++) {
374                     //if stage is part of race
375                     if (stageList.get(j).getRaceID() == raceId) {
376                         try {
377                             removeStageById(stageList.get(j).getStageID());
378                         }
379                         catch (IDNotRecognisedException e) {
380                             System.out.println("ID not recognised");
381                         }
382                     }
383                 }
384                 //remove race
385                 raceList.remove(i);
386             }
387         }
388         assert (checkRaceID(raceId) == false);
389
390     }
391
392     /**
393      * Gets number of stages of a race
394      * @param raceId
395      * @return numStages
396      */
397     @Override
398     public int getNumberOfStages(int raceId) throws IDNotRecognisedException {
399
400         int numStages = 0;
401
402         //if race doesn't exist
403         if (checkRaceID(raceId) == false) {
404             //throw exception
405             throw new IDNotRecognisedException();
406         }
407
408         //for every race
409         for (int i = 0; i < raceList.size(); i++) {
410             //if race exists
411             if (raceList.get(i).getRaceID() == raceId) {
412                 //set numStages to number of stages
413                 numStages = raceList.get(i).getNumOfStages();
414             }
```

```
415         }
416
417         return numStages;
418     }
419
420     /**
421      * Creates a stage and adds it to a race
422      * @param raceId
423      * @param stageName
424      * @param description
425      * @param length
426      * @param startTime
427      * @param type
428      * @return ID of created stage
429      */
430     @Override
431     public int addStageToRace(int raceId, String stageName, String description, double length, LocalDateTime
          startTime,
432             StageType type)
433             throws IDNotRecognisedException, IllegalNameException, InvalidNameException, InvalidLengthException
                {
434
435
436         //for every stage
437         for(int i = 0;i < stageList.size(); i++) {
438             //if stage exists
439             if (stageList.get(i).getStageName() == stageName) {
440                 //throw exception
441                 throw new IllegalNameException();
442             }
443         }
444
445         //if name has invalid attributes
446         if((stageName == null) || (stageName == "") || (stageName.length() > 30) || (stageName.contains(" ")))
              {
447             //throw exception
448             throw new InvalidNameException();
449         }
450
451         //if invalid length
452         if (length < 5) {
453             //throw exception
454             throw new InvalidLengthException();
455         }
456
457         //if race doesn't exists
458         if (checkRaceID(raceId) == false) {
459             //throw exception
460             throw new IDNotRecognisedException();
461         }
462
463         //add new stage to stageList
464         stageList.add(new Stage(raceId, stageName, description, length, startTime, type, stageList));
465         //assert new stage exists
466         assert (stageList.get(stageList.size() - 1).getStageName() == stageName);
```

```
467
468         //for every race
469         for (int i = 0; i < raceList.size(); i++) {
470             //if race exists
471             if (raceList.get(i).getRaceID() == raceId) {
472                 //set number of stages
473                 raceList.get(i).setNumOfStages(stageList);
474             }
475         }
476         //return stageID
477         return stageList.get(stageList.size() - 1).getStageID();
478     }
479
480     /**
481      * Gets the stage IDs of a race
482      * @param raceId
483      * @return raceStageIDList
484      */
485     @Override
486     public int[] getRaceStages(int raceId) throws IDNotRecognisedException {
487
488         int counter = 0;
489         int arraySize = 0;
490
491         //if race doesn't exist
492         if (checkRaceID(raceId) == false){
493             //throw exception
494             throw new IDNotRecognisedException();
495         }
496
497         //for every stage
498         for (int i = 0; i < stageList.size(); i++) {
499             //if stage exists
500             if (stageList.get(i).getRaceID() == raceId) {
501                 //incremenet arraySize
502                 arraySize ++;
503             }
504         }
505
506         int [] raceStageIDList = new int[arraySize];
507
508         //if array contains values
509         if (stageList.size() > 0 ) {
510             //for every stage
511             for(int i = 0; i < stageList.size(); i++) {
512                 //if stage exists
513                 if (stageList.get(i).getRaceID() == raceId) {
514                     //add stageID to raceStageIDList
515                     raceStageIDList[counter] = stageList.get(i).getStageID();
516                     counter ++;
517                 }
518             }
519
520         }
521
```

```
522         return raceStageIDList;
523     }
524 
525     /**
526      * Gets the length of a stage
527      * @param stageId
528      * @return stageLength
529      */
530     @Override
531     public double getStageLength(int stageId) throws IDNotRecognisedException {
532 
533         double stageLength = 0;
534 
535         //if stage doesn't exist
536         if (checkStageID(stageId) == false) {
537             //throw exception
538             throw new IDNotRecognisedException();
539         }
540 
541         //for every stage
542         for (int i = 0; i < stageList.size(); i++) {
543             //if stage exists
544             if (stageList.get(i).getStageID() == stageId) {
545                 //set stageLength to length of stage
546                 stageLength = stageList.get(i).getStageLength();
547             }
548         }
549 
550 
551         return stageLength;
552     }
553 
554     /**
555      * Removes a stage
556      * @param stageId
557      */
558     @Override
559     public void removeStageById(int stageId) throws IDNotRecognisedException {
560 
561         //if stage doesn't exist
562         if (checkStageID(stageId) == false) {
563             //throw exception
564             throw new IDNotRecognisedException();
565         }
566 
567         //for every stage
568         for (int i = 0; i < stageList.size(); i++) {
569             //if stage exists
570             if (stageList.get(i).getStageID() == stageId) {
571 
572                 //for every segment
573                 for (int j = 0; j < segmentList.size(); j++) {
574                     //if segment is part of stage
575                     if (segmentList.get(j).getStageID() == stageId) {
576                         //remove segment
```

```
                try {
                    removeSegment(segmentList.get(j).getSegmentID());
                } catch (InvalidStageStateException e) {
                    System.out.println("Invalid Stage State");
                }
            }
        }

        //for every result
        for (int j = 0; j < resultList.size(); j++) {
            //if result is part of stage
            if (resultList.get(j).getStageID() == stageId) {
                //remove result
                resultList.remove(j);
            }
        }

        //remove stage
        stageList.remove(i);
      }
    }
    assert (checkStageID(stageId) == false);
  }

  /**
   * Adds catergorized climb
   * @param stageId
   * @param location
   * @param type
   * @param averageGradient
   * @param length
   * @return ID of created segment
   */
  @Override
  public int addCategorizedClimbToStage(int stageId, Double location, SegmentType type, Double
      averageGradient,
       Double length) throws IDNotRecognisedException, InvalidLocationException,
           InvalidStageStateException,
       InvalidStageTypeException {

    //if stage doesn't exists
    if (checkStageID(stageId) == false){
      //throw exception
      throw new IDNotRecognisedException();
    }

    //if invalid stageType
    if (checkStageType(stageId) == false) {
      //throw exception
      throw new InvalidStageTypeException();
    }

    //if invalid location
    if (checkSegmentLocation(stageId, location) == false) {
      //throw exception
```

```
630             throw new InvalidLocationException();
631         }
632 
633         //if invalid state
634         if (checkStageState(stageId) == false) {
635             //throw exception
636             throw new InvalidStageStateException();
637         }
638 
639         //add new segment to segmentList
640         segmentList.add(new Segment(stageId, location, type, averageGradient, length, segmentList));
641         //assert segment added
642         assert (segmentList.get(segmentList.size() - 1).getSegmentLocation() == location);
643         //return id of created segment
644         return segmentList.get(segmentList.size() - 1).getSegmentID();
645     }
646 
647 
648     /**
649      * Adds intermediate sprint
650      * @param stageId
651      * @param location
652      * @return ID of created segment
653      */
654     @Override
655     public int addIntermediateSprintToStage(int stageId, double location) throws IDNotRecognisedException,
656             InvalidLocationException, InvalidStageStateException, InvalidStageTypeException {
657 
658         //if stage doesn't exist
659         if (checkStageID(stageId) == false) {
660             //throw exception
661             throw new IDNotRecognisedException();
662         }
663 
664         //if invalid location
665         if (checkSegmentLocation(stageId, location) == false) {
666             //throw exception
667             throw new InvalidLocationException();
668         }
669 
670         //if invalid stageType
671         if (checkStageType(stageId) == false) {
672             //throw exception
673             throw new InvalidStageTypeException();
674         }
675 
676         //if invalid state
677         if (checkStageState(stageId) == false) {
678             //throw exception
679             throw new InvalidStageStateException();
680         }
681 
682         //add new segment to segmentList
683         segmentList.add(new Segment(stageId, location, segmentList));
684         //assert sprint added
```

```
685         assert (segmentList.get(segmentList.size() - 1).getSegmentLocation() == location);
686         //return id of created segment
687         return segmentList.get(segmentList.size() - 1).getSegmentID();
688     }
689 
690     /**
691      * Removes a segment
692      * @param segmentId
693      */
694     @Override
695     public void removeSegment(int segmentId) throws IDNotRecognisedException, InvalidStageStateException {
696 
697         //if segment doesn't exist
698         if (checkSegmentID(segmentId) == false) {
699             throw new IDNotRecognisedException();
700         }
701 
702         for (int i = 0; i < segmentList.size(); i++) {
703             if (segmentList.get(i).getSegmentID() == segmentId) {
704                 //if invalid state
705                 if (checkStageState(segmentList.get(i).getStageID()) == false) {
706                     //throw exception
707                     throw new InvalidStageStateException();
708                 }
709             }
710         }
711 
712         //for every segment
713         for (int i = 0; i < segmentList.size(); i++) {
714             //if segment exists
715             if (segmentList.get(i).getSegmentID() == segmentId) {
716                 //remove segment
717                 segmentList.remove(i);
718             }
719         }
720         assert (checkSegmentID(segmentId) == false);
721     }
722 
723     /**
724      * Changes state of Stage
725      * @param stageId
726      */
727     @Override
728     public void concludeStagePreparation(int stageId) throws IDNotRecognisedException,
        InvalidStageStateException {
729 
730         //if stage doesn't exist
731         if (checkStageID(stageId) == false) {
732             //throw exception
733             throw new IDNotRecognisedException();
734         }
735 
736         //if invalid state
737         if (checkStageState(stageId) == false) {
738             //throw exception
```

```
739             throw new InvalidStageStateException();
740         }
741 
742         //for every stage
743         for (int i = 0; i < stageList.size(); i++) {
744             //if stage exists
745             if (stageList.get(i).getStageID() == stageId) {
746                 //change stage state
747                 stageList.get(i).setStageState("waiting for results");
748             }
749         }
750 
751     }
752 
753     /**
754      * Gets a list of all segment IDs in a stage
755      * @param stageId
756      * @return stageSegmentIDList
757      */
758     @Override
759     public int[] getStageSegments(int stageId) throws IDNotRecognisedException {
760 
761         //if stage doesn't exist
762         if (checkStageID(stageId) == false) {
763             //throw exception
764             throw new IDNotRecognisedException();
765         }
766 
767         int counter = 0;
768         int arraySize = 0;
769 
770         //for every segment
771         for (int i = 0; i < segmentList.size(); i++) {
772             //if segment is part of stage
773             if (segmentList.get(i).getStageID() == stageId) {
774                 //increment array size
775                 arraySize ++;
776             }
777         }
778 
779         //create correct length array
780         int [] stageSegmentIDList = new int[arraySize];
781         double[] stageSegmentLocations = new double[arraySize];
782 
783         //if a segment exists
784         if (segmentList.size() > 0 ) {
785             //for every segment
786             for(int i = 0;i < segmentList.size(); i++) {
787                 //if rider is part of team
788                 if (segmentList.get(i).getStageID() == stageId) {
789                     //add rider id to array
790                     stageSegmentIDList[counter] = segmentList.get(i).getSegmentID();
791                     stageSegmentLocations[counter] = segmentList.get(i).getSegmentLocation();
792                     counter ++;
793                 }
```

```
794         }
795       }
796
797       int n = stageSegmentLocations.length;
798         for (int i = 0; i < n-1; i++) {
799             for (int j = 0; j < n-i-1; j++) {
800                 if (stageSegmentLocations[j] > stageSegmentLocations[j+1])
801                 {
802                     double temp = stageSegmentLocations[j];
803                     stageSegmentLocations[j] = stageSegmentLocations[j+1];
804                     stageSegmentLocations[j+1] = temp;
805
806                     int temp2 = stageSegmentIDList[j];
807                     stageSegmentIDList[j] = stageSegmentIDList[j+1];
808                     stageSegmentIDList[j+1] = temp2;
809                 }
810         }
811       }
812
813       return stageSegmentIDList;
814     }
815
816     /**
817      * Creates a team
818      * @param name
819      * @param description
820      * @return ID of created team
821      */
822     @Override
823     public int createTeam(String name, String description) throws IllegalNameException, InvalidNameException
          {
824
825       //for every team
826       for(int i = 0;i < teamList.size(); i++) {
827         //if team name exists
828         if (teamList.get(i).getTeamName() == name) {
829           //throw exception
830           throw new IllegalNameException();
831         }
832        }
833
834       //if team name has invalid attributes
835       if((name == null) || (name == "") || (name.length() > 30) || (name.contains(" "))) {
836         //throw exception
837         throw new InvalidNameException();
838       }
839
840       //add team to teamList
841       teamList.add(new Team(name, description, teamList));
842       //assert team added
843       assert (teamList.get(teamList.size() - 1).getTeamName() == name);
844       //return ID of created team
845       return teamList.get(teamList.size() - 1).getTeamID();
846
847     }
```

```
848
849     /**
850      * Removes a team
851      * @param teamId
852      */
853     @Override
854     public void removeTeam(int teamId) throws IDNotRecognisedException {
855
856        //if team doesn't exist
857        if (checkTeamID(teamId) == false) {
858           //throw exception
859           throw new IDNotRecognisedException();
860        }
861
862        //array of riderIDs to remove
863        int[] ridersToRemove = getTeamRiders(teamId);
864
865        //for every rider to remove
866        for (int i = 0; i < ridersToRemove.length; i++) {
867           //remove rider
868           removeRider(ridersToRemove[i]);
869        }
870
871        //for every team
872        for (int i = 0; i < teamList.size(); i++) {
873           //if team exists
874           if (teamList.get(i).getTeamID() == teamId) {
875              teamList.remove(i);
876           }
877        }
878
879        assert (checkTeamID(teamId) == false);
880     }
881
882     /**
883      * Gets IDs of all teams
884      * @return teamIDList
885      */
886     @Override
887     public int[] getTeams() {
888
889        int [] teamIDList = new int[teamList.size()];
890        if (teamList.size() > 0 ) {
891           for(int i = 0;i < teamList.size(); i++) {
892              teamIDList[i]= teamList.get(i).getTeamID();
893           }
894           assert teamIDList[0] == teamList.get(0).getTeamID();
895        }
896
897        return teamIDList;
898     }
899
900     /**
901      * Get IDs of riders of a team
902      * @param teamId
```

```
903      * @return teamRiderIDList
904      */
905     @Override
906     public int[] getTeamRiders(int teamId) throws IDNotRecognisedException {
907 
908        //if team doesn't exist
909        if (checkTeamID(teamId) == false) {
910           //throw exception
911           throw new IDNotRecognisedException();
912        }
913 
914        int counter = 0;
915        int arraySize = 0;
916 
917        //for every rider
918        for (int i = 0; i < riderList.size(); i++) {
919           //if rider is part of team
920           if (riderList.get(i).getTeamID() == teamId) {
921              //increment array size
922              arraySize ++;
923           }
924        }
925 
926        //create correct length array
927        int [] teamRiderIDList = new int[arraySize];
928 
929        //if a rider exists
930        if (riderList.size() > 0 ) {
931           //for every rider
932           for(int i = 0;i < riderList.size(); i++) {
933              //if rider is part of team
934              if (riderList.get(i).getTeamID() == teamId) {
935                 //add rider id to array
936                 teamRiderIDList[counter] = riderList.get(i).getRiderID();
937                 counter ++;
938              }
939           }
940 
941        }
942 
943        return teamRiderIDList;
944     }
945 
946     /**
947      * Creates a rider
948      * @param name
949      * @param teamID
950      * @param yearOfBirth
951      * @return riderID
952      */
953     @Override
954     public int createRider(int teamID, String name, int yearOfBirth) throws IDNotRecognisedException,
         IllegalArgumentException {
955 
956        //if name or year of birth is invalid
```

```
957         if ((name == null) || (yearOfBirth < 1900)) {
958             //throw exception
959             throw new IllegalArgumentException();
960         }
961 
962         //if team doesn't exist
963         if(checkTeamID(teamID) == false) {
964             //throw exception
965             throw new IDNotRecognisedException();
966         }
967 
968         //add created rider to riderList
969         riderList.add(new Rider(teamID, yearOfBirth, name, riderList));
970         //return ID of created rider
971         return riderList.get(riderList.size() - 1).getRiderID();
972 
973     }
974 
975     /**
976      * Removes a rider
977      * @param riderId
978      */
979     @Override
980     public void removeRider(int riderId) throws IDNotRecognisedException {
981 
982         //if rider doesn't exist
983         if (checkRiderID(riderId) == false) {
984             //throw exception
985                 throw new IDNotRecognisedException();
986             }
987 
988         //for every result
989         for (int i = 0; i < resultList.size(); i++) {
990             //if result is part of rider
991             if (resultList.get(i).getRiderID() == riderId) {
992                 //remove result
993                 resultList.remove(i);
994             }
995         }
996 
997         //for every rider
998         for (int i = 0; i < riderList.size(); i++) {
999             //if rider exists
1000            if (riderList.get(i).getRiderID() == riderId) {
1001                //remove rider
1002                riderList.remove(i);
1003            }
1004        }
1005    }
1006 
1007    /**
1008     * Adds results to a rider in a stage
1009     * @param stageId
1010     * @param riderId
1011     * @param checkpoints
```

```
1012      */
1013     @Override
1014     public void registerRiderResultsInStage(int stageId, int riderId, LocalTime... checkpoints)
1015           throws IDNotRecognisedException, DuplicatedResultException, InvalidCheckpointsException,
1016           InvalidStageStateException {
1017
1018           //if stage doesn't exist
1019           if (checkStageID(stageId) == false) {
1020              //throw exception
1021              throw new IDNotRecognisedException();
1022           }
1023
1024           //if rider doesn't exist
1025           if (checkRiderID(riderId) == false) {
1026              //throw exception
1027              throw new IDNotRecognisedException();
1028           }
1029
1030           //if stage state is invalid
1031           if (checkStageState(stageId) == true) {
1032              //throw exception
1033              throw new InvalidStageStateException();
1034           }
1035
1036           int numberOfSegments = 0;
1037           //for every segment
1038           for (int i = 0; i < segmentList.size(); i++) {
1039              //if segment is part of stage
1040              if (segmentList.get(i).getStageID() == stageId) {
1041                 //increment number of stages
1042                 numberOfSegments++;
1043              }
1044           }
1045
1046           //if number of checkpoints != number of segments + 2
1047           if (checkpoints.length != numberOfSegments + 2) {
1048              //throw exception
1049              throw new InvalidCheckpointsException();
1050           }
1051
1052           //for every result
1053           for (int i = 0; i < resultList.size(); i++) {
1054              //if result is part of rider and stage
1055              if ((resultList.get(i).getRiderID() == riderId) &&
1056              (resultList.get(i).getStageID() == stageId)) {
1057                 //throw exception
1058                 throw new DuplicatedResultException();
1059              }
1060           }
1061
1062        //add created result to resultList
1063        resultList.add(new Result(stageId, riderId, checkpoints, resultList));
1064
1065     }
1066
```

```
1067    /**
1068     * Gets segment times and elapsed time for rider
1069     * @param stageId
1070     * @param riderId
1071     * @return segmentTimes
1072     */
1073    @Override
1074    public LocalTime[] getRiderResultsInStage(int stageId, int riderId) throws IDNotRecognisedException {
1075
1076       //if rider or stage doesn't exist
1077       if ((checkRiderID(riderId) == false) || (checkStageID(stageId) == false)) {
1078          //throw exception
1079          throw new IDNotRecognisedException();
1080       }
1081
1082       int arraySize = 0;
1083       //for every result
1084       for (int i = 0; i < resultList.size(); i++) {
1085          //if result is part of rider and stage
1086          if ((resultList.get(i).getRiderID() == riderId) &&
1087          (resultList.get(i).getStageID() == stageId)) {
1088             arraySize = resultList.get(i).getCheckpoints().length;
1089          }
1090       }
1091
1092       LocalTime[] segmentTimes = new LocalTime[arraySize + 1];
1093
1094       //for every result
1095       for (int i = 0; i < resultList.size(); i++) {
1096          //if result is part of rider and stage
1097          if ((resultList.get(i).getRiderID() == riderId) &&
1098          (resultList.get(i).getStageID() == stageId)) {
1099             segmentTimes = resultList.get(i).getCheckpoints();
1100          }
1101       }
1102
1103       //append total elapsed time to array
1104       segmentTimes[segmentTimes.length - 1] = getTotalTime(riderId, stageId);
1105
1106       return segmentTimes;
1107    }
1108
1109    /**
1110     * Gets adjusted time for rider in stage
1111     * @param riderId
1112     * @param stageId
1113     * @return riderAdjustedTime
1114     */
1115    @Override
1116    public LocalTime getRiderAdjustedElapsedTimeInStage(int stageId, int riderId) throws
         IDNotRecognisedException {
1117
1118       //if rider doesn't exist
1119       if ((checkRiderID(riderId) == false) || checkStageID(stageId) == false) {
1120          //throw exception
```

```
            throw new IDNotRecognisedException();
        }

        LocalTime riderAdjustedTime = null;

        LocalTime[] riderTimesList = getRankedAdjustedElapsedTimesInStage(stageId);
        int[] riderRankList = getRidersRankInStage(stageId);

        //loop through IDs
        for (int i = 0; i < riderRankList.length; i++) {
            //if rider ID matches
            if (riderRankList[i] == riderId) {
                riderAdjustedTime = riderTimesList[i];
            }
        }
        return riderAdjustedTime;
    }

    /**
     * Removes rider results from stage
     * @param riderId
     * @param stageId
     */
    @Override
    public void deleteRiderResultsInStage(int stageId, int riderId) throws IDNotRecognisedException {

        //if rider doesn't exist
        if (checkRiderID(riderId) == false) {
            //throw exception
            throw new IDNotRecognisedException();
        }

        //for every result
        for (int i = 0; i < resultList.size(); i++) {
            //if result is part of rider
            if ((resultList.get(i).getRiderID() == riderId) &&
            (resultList.get(i).getStageID() == stageId)) {
                //remove result
                resultList.remove(i);
            }
        }

    }

    /**
     * Gets list of rider IDs sorted by their elapsed time
     * @param stageId
     * @return riderRank
     */
    @Override
    public int[] getRidersRankInStage(int stageId) throws IDNotRecognisedException {

        //if stage doesn't exist
        if (checkStageID(stageId) == false) {
            //throw exception
```

```
            throw new IDNotRecognisedException();
        }

        int arraySize = 0;

        //for every result
        for (int i = 0; i < resultList.size(); i++) {
            //if result is part of stage
            if (resultList.get(i).getStageID() == stageId) {
                //increment arraySize
                arraySize++;
            }
        }


        int[] riderRank = new int[arraySize];
        LocalTime[] riderTimes = new LocalTime[arraySize];
        int count = 0;

        //for every result
        for (int i = 0; i < resultList.size(); i++) {
            //if result is part of stage
            if (resultList.get(i).getStageID() == stageId) {
                //add rider ID to riderRank array
                riderRank[count] = resultList.get(i).getRiderID();
                riderTimes[count] = getTotalTime(resultList.get(i).getRiderID(), stageId);
                count++;
            }
        }

        //sort by total elapsed time
          int n = riderTimes.length;
          for (int i = 0; i < n-1; i++) {
              for (int j = 0; j < n-i-1; j++) {
                  if (riderTimes[j].compareTo(riderTimes[j + 1]) > 0) {
                      // swap arr[j+1] and arr[j]
                      LocalTime temp = riderTimes[j];
                      riderTimes[j] = riderTimes[j+1];
                      riderTimes[j+1] = temp;

                      int mainTemp = riderRank[j];
                      riderRank[j] = riderRank[j + 1];
                      riderRank[j + 1] = mainTemp;
                  }
              }
          }
          return riderRank;
    }

    /**
     * Gets array of ranked adjusted times in stage
     * @param stageId
     * @return riderTimesArray
     */
    @Override
```

```
1231    public LocalTime[] getRankedAdjustedElapsedTimesInStage(int stageId) throws IDNotRecognisedException {
1232
1233       //if stage doesn't exist
1234       if (checkStageID(stageId) == false) {
1235          //throw exception
1236          throw new IDNotRecognisedException();
1237       }
1238
1239       int[] riderRankArray = getRidersRankInStage(stageId);
1240       LocalTime[] riderTimesArray = new LocalTime [riderRankArray.length];
1241
1242       //for every rider in array
1243       if (riderTimesArray.length > 0) {
1244          for (int i = 0; i < riderRankArray.length; i++) {
1245             riderTimesArray[i] = getTotalTime(riderRankArray[i], stageId);
1246          }
1247
1248          //for every time in array
1249          for (int i = riderTimesArray.length - 1; i <= 1; i--) {
1250             if (riderTimesArray[i - 1].until(riderTimesArray[i], ChronoUnit.MILLIS) < 1000) {
1251                riderRankArray[i] = riderRankArray[i - 1];
1252             }
1253          }
1254       }
1255
1256       return riderTimesArray;
1257    }
1258
1259    /**
1260     * Gets the number of points obtained by riders in stage
1261     * @param stageId
1262     * @return riderIDList
1263     */
1264    @Override
1265    public int[] getRidersPointsInStage(int stageId) throws IDNotRecognisedException {
1266
1267       //if stage doesn't exist
1268       if (checkStageID(stageId) == false) {
1269          //throw exception
1270          throw new IDNotRecognisedException();
1271       }
1272
1273       int[] riderIDList = getRidersRankInStage(stageId);
1274       int[] riderPoints = new int[riderIDList.length];
1275
1276       //for every stage
1277       for (int i = 0; i < stageList.size(); i++) {
1278          if (stageList.get(i).getStageID() == stageId) {
1279             if (stageList.get(i).getStageType() == StageType.FLAT) {
1280                for (int j = 0; j < riderIDList.length; j++) {
1281                   switch (j) {
1282                      case 0:
1283                         riderPoints[j] = 50;
1284                         break;
1285                      case 1:
```

```
                    riderPoints[j] = 30;
                    break;
                case 2:
                    riderPoints[j] = 20;
                    break;
                case 3:
                    riderPoints[j] = 18;
                    break;
                case 4:
                    riderPoints[j] = 16;
                    break;
                case 5:
                    riderPoints[j] = 14;
                    break;
                case 6:
                    riderPoints[j] = 12;
                    break;
                case 7:
                    riderPoints[j] = 10;
                    break;
                case 8:
                    riderPoints[j] = 8;
                    break;
                case 9:
                    riderPoints[j] = 7;
                    break;
                case 10:
                    riderPoints[j] = 6;
                    break;
                case 11:
                    riderPoints[j] = 5;
                    break;
                case 12:
                    riderPoints[j] = 4;
                    break;
                case 13:
                    riderPoints[j] = 3;
                    break;
                case 14:
                    riderPoints[j] = 2;
                    break;
                default:
                    riderIDList[j] = 0;
                    break;
            }
        }
    }
    else if (stageList.get(i).getStageType() == StageType.MEDIUM_MOUNTAIN) {
        for (int j = 0; j < riderIDList.length; j++) {
            switch (j) {
                case 0:
                    riderPoints[j] = 30;
                    break;
                case 1:
                    riderPoints[j] = 25;
```

```
                    break;
                case 2:
                    riderPoints[j] = 22;
                    break;
                case 3:
                    riderPoints[j] = 19;
                    break;
                case 4:
                    riderPoints[j] = 17;
                    break;
                case 5:
                    riderPoints[j] = 15;
                    break;
                case 6:
                    riderPoints[j] = 13;
                    break;
                case 7:
                    riderPoints[j] = 11;
                    break;
                case 8:
                    riderPoints[j] = 9;
                    break;
                case 9:
                    riderPoints[j] = 7;
                    break;
                case 10:
                    riderPoints[j] = 6;
                    break;
                case 11:
                    riderPoints[j] = 5;
                    break;
                case 12:
                    riderPoints[j] = 4;
                    break;
                case 13:
                    riderPoints[j] = 3;
                    break;
                case 14:
                    riderPoints[j] = 2;
                    break;
                default:
                    riderIDList[j] = 0;
                    break;
            }
        }
    }
    else if ((stageList.get(i).getStageType() == StageType.HIGH_MOUNTAIN)
    || stageList.get(i).getStageType() == StageType.TT) {
        for (int j = 0; j < riderIDList.length; j++) {
            switch (j) {
                case 0:
                    riderPoints[j] = 20;
                    break;
                case 1:
                    riderPoints[j] = 17;
```

```
                        break;
                    case 2:
                        riderPoints[j] = 15;
                        break;
                    case 3:
                        riderPoints[j] = 13;
                        break;
                    case 4:
                        riderPoints[j] = 11;
                        break;
                    case 5:
                        riderPoints[j] = 10;
                        break;
                    case 6:
                        riderPoints[j] = 9;
                        break;
                    case 7:
                        riderPoints[j] = 8;
                        break;
                    case 8:
                        riderPoints[j] = 7;
                        break;
                    case 9:
                        riderPoints[j] = 6;
                        break;
                    case 10:
                        riderPoints[j] = 5;
                        break;
                    case 11:
                        riderPoints[j] = 4;
                        break;
                    case 12:
                        riderPoints[j] = 3;
                        break;
                    case 13:
                        riderPoints[j] = 2;
                        break;
                    case 14:
                        riderPoints[j] = 1;
                        break;
                    default:
                        riderIDList[j] = 0;
                        break;
                }
            }
        }
    }
    return riderIDList;
}

/**
 * Gets riders mountain points in stage
 * @param stageId
 */
```

```
1451    @Override
1452    public int[] getRidersMountainPointsInStage(int stageId) throws IDNotRecognisedException {
1453
1454       return null;
1455    }
1456
1457    /**
1458     * Erases the cycling portal objects
1459     */
1460    @Override
1461    public void eraseCyclingPortal() {
1462       riderList.clear();
1463       raceList.clear();
1464       stageList.clear();
1465       segmentList.clear();
1466       resultList.clear();
1467       teamList.clear();
1468    }
1469
1470    /**
1471     * Saves cycling portal objects to file
1472     * @param filename
1473     */
1474    @Override
1475    public void saveCyclingPortal(String filename) throws IOException {
1476       try {
1477          ObjectOutputStream out = new ObjectOutputStream(
1478             new FileOutputStream(filename));
1479          out.writeObject(riderList);
1480          out.writeObject(raceList);
1481          out.writeObject(stageList);
1482          out.writeObject(segmentList);
1483          out.writeObject(resultList);
1484          out.writeObject(teamList);
1485          out.close();
1486       }
1487       catch(IOException e) {
1488          System.out.println(e.getMessage());
1489       }
1490
1491
1492    }
1493
1494    /**
1495     * Loads cycling portal objects from file
1496     * @param filename
1497     */
1498    @Override
1499    @SuppressWarnings("unchecked")
1500    public void loadCyclingPortal(String filename) throws IOException, ClassNotFoundException {
1501
1502       try {
1503          ObjectInputStream in = new ObjectInputStream(new FileInputStream(
1504             filename));
1505          riderList = (ArrayList<Rider>) in.readObject();
```

```
1506            raceList = (ArrayList<Race>)in.readObject();
1507            stageList = (ArrayList<Stage>)in.readObject();
1508            segmentList = (ArrayList<Segment>)in.readObject();
1509            resultList = (ArrayList<Result>)in.readObject();
1510            teamList = (ArrayList<Team>)in.readObject();
1511            in.close();
1512        }
1513        catch(IOException e) {
1514            System.out.println(e.getMessage());
1515        }
1516        catch(ClassNotFoundException e) {
1517            System.out.println(e.getMessage());
1518        }
1519        catch (ClassCastException e) {
1520            System.out.println(e.getMessage());
1521        }
1522
1523    }
1524
1525    /**
1526     * Removes a race
1527     * @param name
1528     */
1529    @Override
1530    public void removeRaceByName(String name) throws NameNotRecognisedException{
1531
1532        //if race doesn't exist
1533        if (checkRaceName(name) == false) {
1534            //throw exception
1535            throw new NameNotRecognisedException();
1536        }
1537
1538        //for every race
1539        for (int i = 0; i < raceList.size(); i++) {
1540            //if race exists
1541            if (raceList.get(i).getRaceName() == name) {
1542                //remove stages
1543                int raceToRemove = raceList.get(i).getRaceID();
1544                //for every stage
1545                for (int j = 0; j < stageList.size(); j++) {
1546                    //if stage is part of race
1547                    if (stageList.get(j).getRaceID() == raceToRemove) {
1548                        try {
1549                            removeStageById(stageList.get(j).getStageID());
1550                        }
1551                        catch (IDNotRecognisedException e) {
1552                            System.out.println("ID not recognised");
1553                        }
1554                    }
1555                }
1556                //remove race
1557                raceList.remove(i);
1558            }
1559        }
1560        assert (checkRaceName(name) == false);
```

```
1561
1562    }
1563
1564    @Override
1565    public LocalTime[] getGeneralClassificationTimesInRace(int raceId) throws IDNotRecognisedException {
1566       LocalTime[] classTimes = new LocalTime[getRidersRankInStage(getRaceStages(raceId)[0]).length];
1567
1568       int[] classRiders = getRidersRankInStage(getRaceStages(raceId)[0]);
1569
1570       LocalTime riderAdjTime = null;
1571
1572       //for every stage in race
1573       int[] raceStages = getRaceStages(raceId);
1574
1575       for (int i = 0; i < raceStages.length; i++) {
1576          int[] stageRiders = getRidersRankInStage(i);
1577
1578          //for eveyr rider in stage
1579          for (int j = 0; j < stageRiders.length; j++) {
1580
1581             riderAdjTime = getRiderAdjustedElapsedTimeInStage(raceStages[i], stageRiders[j]);
1582
1583             //loop through riders array
1584             for (int k = 0; k < classRiders.length; k++) {
1585                //if rider ID matches
1586                if (stageRiders[j] == classRiders[k]) {
1587                   classTimes[k] = classTimes[k].plusHours(riderAdjTime.getHour()).plusMinutes(
1588                      riderAdjTime.getMinute()).plusSeconds(riderAdjTime.getSecond()).plusNanos(
1589                         riderAdjTime.getNano());
1590                }
1591             }
1592          }
1593
1594       }
1595
1596       //sort array
1597       int n = classRiders.length;
1598         for (int i = 0; i < n-1; i++)
1599             for (int j = 0; j < n-i-1; j++)
1600                 if (classRiders[j] > classRiders[j+1])
1601                 {
1602                     int temp = classRiders[j];
1603                     classRiders[j] = classRiders[j+1];
1604                     classRiders[j+1] = temp;
1605
1606                     LocalTime temp2 = classTimes[j];
1607                     classTimes[j] = classTimes[j+1];
1608                     classTimes[j+1] = temp2;
1609                 }
1610
1611       return classTimes;
1612    }
1613
1614    /**
1615     * Get the overall points of riders in race
```

```
1616      * @param raceId
1617      * @return
1618      */
1619     @Override
1620     public int[] getRidersPointsInRace(int raceId) throws IDNotRecognisedException {
1621
1622        //if race doesn't exist
1623        if (checkRaceID(raceId) == false) {
1624           //throw exception
1625           throw new IDNotRecognisedException();
1626        }
1627
1628        //for every stage
1629        for (int i = 0; i < stageList.size(); i++) {
1630           //if stage is part of race
1631           if (stageList.get(i).getRaceID() == raceId) {
1632
1633           }
1634        }
1635        return null;
1636     }
1637
1638     @Override
1639     public int[] getRidersMountainPointsInRace(int raceId) throws IDNotRecognisedException {
1640
1641
1642        return null;
1643     }
1644
1645     @Override
1646     public int[] getRidersGeneralClassificationRank(int raceId) throws IDNotRecognisedException {
1647
1648        LocalTime[] classTimes = new LocalTime[getRidersRankInStage(getRaceStages(raceId)[0]).length];
1649
1650        int[] classRiders = getRidersRankInStage(getRaceStages(raceId)[0]);
1651
1652        LocalTime riderAdjTime = null;
1653
1654        //for every stage in race
1655        int[] raceStages = getRaceStages(raceId);
1656
1657        for (int i = 0; i < raceStages.length; i++) {
1658           int[] stageRiders = getRidersRankInStage(i);
1659
1660           //for eveyr rider in stage
1661           for (int j = 0; j < stageRiders.length; j++) {
1662
1663              riderAdjTime = getRiderAdjustedElapsedTimeInStage(raceStages[i], stageRiders[j]);
1664
1665              //loop through riders array
1666              for (int k = 0; k < classRiders.length; k++) {
1667                 //if rider ID matches
1668                 if (stageRiders[j] == classRiders[k]) {
1669                    classTimes[k] = classTimes[k].plusHours(riderAdjTime.getHour()).plusMinutes(
1670                       riderAdjTime.getMinute()).plusSeconds(riderAdjTime.getSecond()).plusNanos(
```

```
                        riderAdjTime.getNano());
                }
            }
        }

    }

    //sort array
    int n = classRiders.length;
      for (int i = 0; i < n-1; i++)
          for (int j = 0; j < n-i-1; j++)
              if (classRiders[j] > classRiders[j+1])
              {
                  int temp = classRiders[j];
                  classRiders[j] = classRiders[j+1];
                  classRiders[j+1] = temp;

                  LocalTime temp2 = classTimes[j];
                  classTimes[j] = classTimes[j+1];
                  classTimes[j+1] = temp2;
              }

    return classRiders;
  }

  @Override
  public int[] getRidersPointClassificationRank(int raceId) throws IDNotRecognisedException {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public int[] getRidersMountainPointClassificationRank(int raceId) throws IDNotRecognisedException {
    // TODO Auto-generated method stub
    return null;
  }

}
```

8 CyclingPortalInterfaceTestApp.java

```

import java.time.LocalDate;
import java.time.LocalDateTime;

import cycling.CyclingPortal;
import cycling.CyclingPortalInterface;
import cycling.IDNotRecognisedException;
import cycling.IllegalNameException;
import cycling.InvalidLengthException;
import cycling.InvalidLocationException;
import cycling.InvalidNameException;
import cycling.InvalidStageStateException;
```

```
14 import cycling.InvalidStageTypeException;
15 import cycling.NameNotRecognisedException;
16 import cycling.SegmentType;
17 import cycling.StageType;
18
19
20 /**
21  * A short program to illustrate an app testing some minimal functionality of a
22  * concrete implementation of the CyclingPortalInterface interface -- note you
23  * will want to increase these checks, and run it on your CyclingPortal class
24  * (not the BadCyclingPortal class).
25  *
26  *
27  * @author Diogo Pacheco
28  * @version 1.0
29  */
30 public class CyclingPortalInterfaceTestApp {
31
32    /**
33     * Test method.
34     *
35     * @param args not used
36     *
37     */
38
39
40    public static void main(String[] args) throws IllegalNameException, InvalidNameException {
41       System.out.println("The system compiled and started the execution...");
42
43       CyclingPortalInterface portal = new CyclingPortal();
44       //CyclingPortalInterface portal = new CyclingPortal();
45
46       //create race
47       assert (portal.createRace("testRace", "testDesc") == 2000);
48
49       //add stage to race
50       try {
51          assert (portal.addStageToRace(2000, "testStage", "testDesc", 5.0, LocalDateTime.of(2022, 3, 29, 10,
                0, 0), StageType.FLAT)) == 2000;
52       } catch (IDNotRecognisedException e1) {
53          e1.printStackTrace();
54       } catch (InvalidLengthException e1) {
55          e1.printStackTrace();
56       }
57
58       //add sprint to stage
59       try {
60          assert (portal.addIntermediateSprintToStage(2000, 5.0) == 2000);
61       } catch (IDNotRecognisedException e1) {
62          e1.printStackTrace();
63       } catch (InvalidLocationException e1) {
64          e1.printStackTrace();
65       } catch (InvalidStageStateException e1) {
66          e1.printStackTrace();
67       } catch (InvalidStageTypeException e1) {
```

```
            e1.printStackTrace();
        }

        //add catergorized climb to stage
        try {
            assert (portal.addCategorizedClimbToStage(2000, 5.0, SegmentType.C4, 5.0, 5.0) == 2001);
        } catch (IDNotRecognisedException e1) {
            e1.printStackTrace();
        } catch (InvalidLocationException e1) {
            e1.printStackTrace();
        } catch (InvalidStageStateException e1) {
            e1.printStackTrace();
        } catch (InvalidStageTypeException e1) {
            e1.printStackTrace();
        }

        //create team
        assert (portal.createTeam("testTeam", "testDesc") == 2000);

        //create rider
        try {
            assert (portal.createRider(2000, "testRider", 2022) == 2000);
        } catch (IllegalArgumentException e1) {
            e1.printStackTrace();
        } catch (IDNotRecognisedException e1) {
            e1.printStackTrace();
        }

        //get number of stages
        try {
            assert (portal.getNumberOfStages(2000) == 1);
        } catch (IDNotRecognisedException e1) {
            e1.printStackTrace();
        }

        //get race stages
        try {
            assert (portal.getRaceStages(2000).length == 1);
        } catch (IDNotRecognisedException e1) {
            e1.printStackTrace();
        }

        //get stage length
        try {
            assert (portal.getStageLength(2000) == 5.0);
        } catch (IDNotRecognisedException e1) {
            e1.printStackTrace();
        }


        //get stage segments
        try {
            assert (portal.getStageSegments(2000).length == 2);
        } catch (IDNotRecognisedException e1) {
            e1.printStackTrace();
```

```
123         }
124
125         //get teams
126         assert (portal.getTeams().length == 1);
127
128         //get team riders
129         try {
130            assert (portal.getTeamRiders(2000).length == 1);
131         } catch (IDNotRecognisedException e1) {
132            e1.printStackTrace();
133         }
134
135         //view race details
136         try {
137            System.out.println(portal.viewRaceDetails(2000));
138         } catch (IDNotRecognisedException e) {
139            e.printStackTrace();
140         }
141
142         //get race IDs
143         assert (portal.getRaceIds()[0] == 2000);
144
145         //remove segment
146         try {
147            portal.removeSegment(2000);
148            portal.removeSegment(2001);
149         } catch (IDNotRecognisedException e1) {
150            e1.printStackTrace();
151         } catch (InvalidStageStateException e1) {
152            e1.printStackTrace();
153         }
154         try {
155            assert (portal.getStageSegments(2000).length == 0);
156         } catch (IDNotRecognisedException e1) {
157            e1.printStackTrace();
158         }
159
160         //remove stage
161         try {
162            portal.removeStageById(2000);
163         } catch (IDNotRecognisedException e1) {
164            e1.printStackTrace();
165         }
166         try {
167            assert (portal.getRaceStages(2000).length == 0);
168         } catch (IDNotRecognisedException e1) {
169            e1.printStackTrace();
170         }
171
172         //remove rider
173         try {
174            portal.removeRider(2000);
175         } catch (IDNotRecognisedException e1) {
176            e1.printStackTrace();
177         }
```

```
178         try {
179            assert (portal.getTeamRiders(2000).length == 0);
180         } catch (IDNotRecognisedException e1) {
181            e1.printStackTrace();
182         }
183
184         //remove team
185         try {
186            portal.removeTeam(2000);
187         } catch (IDNotRecognisedException e1) {
188            e1.printStackTrace();
189         }
190         assert (portal.getTeams().length == 0);
191
192
193         //remove race by ID
194         try {
195            portal.removeRaceById(2000);
196            assert (portal.getRaceIds().length == 0);
197         } catch (IDNotRecognisedException e) {
198            e.printStackTrace();
199         }
200         assert (portal.getRaceIds().length == 0);
201
202         //remove race by name
203         portal.createRace("testRemove", "testDesc");
204
205         try {
206            portal.removeRaceByName("testRemove");
207         } catch (NameNotRecognisedException e) {
208            e.printStackTrace();
209         }
210         assert (portal.getRaceIds().length == 0);
211
212
213      }
214
215   }
```