1 CyclingPortal.java

```java
package cycling;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.HashMap;



public class CyclingPortal implements CyclingPortalInterface {

    ArrayList<Race> Races = new ArrayList<>();
    ArrayList<Stage> Stages = new ArrayList<>();
    ArrayList<Segment> Segments = new ArrayList<>();
    ArrayList<Team> Teams = new ArrayList<>();
    ArrayList<Rider> Riders = new ArrayList<>();


    /**
     * @return int[]
     */
    @Override
    public int[] getRaceIds() {
        int[] idArray = new int[Races.size()];
        for (int i = 0; i < Races.size(); i++) {
            idArray[i] = Races.get(i).getId();
        }
        return idArray;
    }


    /**
     * @param name
     * @param description
     * @return int
     * @throws InvalidNameException
     * @throws IllegalNameException
     */
```

```
53      @Override
54      public int createRace(String name, String description) throws InvalidNameException,
            IllegalNameException {
55          // If the name is null, empty, has more than 30 characters, or has white spaces. Throw
                IllegalNameException
56          if (name == null) { // Checks if name is null
57              throw new InvalidNameException("Name cannot be Null"); // Throws a new InvalidNameException with
                    the message "Name cannot be Null"
58          }
59          if (name.length() > 30) { // Checks if the length of the name is greater than 30
60              throw new InvalidNameException("Name must be less than 30 characters"); // Throws a new
                    InvalidNameException with the message "Name must be less than 30 characters"
61          }
62          if (name.contains(" ")){ //Checks if any white space is present in the provided name
63              throw new InvalidNameException("Name must not contain white space"); // Throws a new
                    InvalidNameException with the message "Name must not contain white space"
64          }
65          // If the name already exists in the platform. Throw IllegalNameException
66          for (Race i:Races) { // For each race in races
67              if (i.getName().equals(name)) { // Checks if the name of the race is equal to the name provided
68                  throw new IllegalNameException("Race name already exists"); // Throws a new
                        IllegalNameException with the message "Race name already exists"
69              }
70          }
71
72          Races.add(new Race(name, description)); // Creates a new race and add it to the Races ArrayList
73          return Races.get(Races.size()-1).getId(); // Returns the ID of the race at the end of the list
74          // return race1.getId();
75      }
76
77
78      /**
79       * @param raceId
80       * @return String
81       * @throws IDNotRecognisedException
82       */
83      @Override
84      public String viewRaceDetails(int raceId) throws IDNotRecognisedException {
85          assert raceId > 0 : "Race ID must be greater than 0"; // Throws an assertion exception if the raceId
                is less than 0
86          Boolean found = false;
87          String description = "", name = "";
88          int numberOfStages = 0;
89          double totalLength = 0;
90          for (Race race : Races) { // Starts a loop that loops through the races in the race array list
91              if (race.getId() == raceId) { // Checks if the ID of the race is equal to the raceID provided
92                  found = true; // Sets the found variable to true
93                  name = race.getName(); // Gets the name of the race
94                  description = race.getDescription(); // Gets the description of the race
95                  numberOfStages = this.getNumberOfStages(raceId); // Gets the nymber of stages in the race
96                  for (int stageID : this.getRaceStages(raceId)) { // Starts a for loop that loops through the
                        stages with the given race ID
97                      for (Stage stage : Stages) { // A for loop that loops through the stages in the stages
                            array list
98                          if (stage.getStageId() == stageID) { // Checks if the stage ID of the stage is equal
```

```
                        to the provided stage ID
                        totalLength += stage.getLength(); // Adds the length of the stage to the total
                            length variable
                    }
                }
            }
            break; // Breaks from the for loop
        }
    }
    if (!found) { // Checks if the found variable is set to false
        throw new IDNotRecognisedException("Race ID doesn't match a race on the system"); // Throws a
            new IDNotRecognised exception
    }
    else{
        return String.format("RaceId : %d, \nName: %s, \nDecription: %s, \nNumber of Stages: %d, \nTotal
            Length: %fKM",raceId,name,description,numberOfStages,totalLength); // Returns a formatted
            strring with all the information about the race
    }
}


/**
 * @param raceId
 * @throws IDNotRecognisedException
 */
@Override
public void removeRaceById(int raceId) throws IDNotRecognisedException {
    assert raceId > 0 : "Race ID must be greater than 0"; // Throws an assertion exception if the raceId
        is less than 0
    boolean found = false; // Declares a variable found and sets it equal to false
    for (Race race : Races){ // Loops through all the races in the race array list
        if (race.getId()==raceId){ // Checks if the race id for that race is equal to the given race id
            found = true; // Sets found equal to true
            for (Stage stage : Stages){ // Loops through all the stages in the stage array list
                if (stage.getRaceID() == raceId){ // Checks if the race id for that stage is equal to the
                    given race id
                    removeStageById(stage.getRaceID()); // Removes the stage with the given race id
                }
            }
            break; // Breaks from the for loop
        }
    }
    if (!found){ // Checks if the found variable is equal to false
        throw new IDNotRecognisedException("That Race Id is not recognised"); // Throws a new
            IDNotRecognisedExcception with message: 'That Race Id is not recognised'
    }
    for (int i=0; i<Races.size(); i++){ // Loops through the following code i number of times, where i
        is the length of the races array list
        if (Races.get(i).getId() == raceId){ // Checks if the race id for that race is equal to the
            given race id
            Races.remove(i); // Removes that race from the system
            break; // Breaks from the for loop
        }
    }

```

```
144     }
145
146
147     /**
148      * @param raceId
149      * @return int
150      * @throws IDNotRecognisedException
151      */
152     @Override
153     public int getNumberOfStages(int raceId) throws IDNotRecognisedException {
154         assert raceId > 0 : "Race ID must be greater than 0"; // Throws an assertion exception if the raceId
                is less than 0
155         boolean found = false; // Defines the found variable as a boolean with the value of False.
156         for (Race race : Races) { // For each Race in races
157             if (race.getId() == raceId) { // Checks if the ID of the race is equal to the race ID
158                 found = true; // Sets found to True
159             }
160         }
161         if (found) { // Checks if found is equal to True
162             int count = 0; // Defines a variable count as an int and sets it to 0
163             for (Stage stage : Stages) { // Starts a loop which loops through the stages in the Stages array
                    list
164                 if (stage.getRaceID() == raceId) { // Checks if the race ID of the stage is equal to the race
                        ID provided
165                     count++; // Adds one to the count variable
166                 }
167             }
168             return count; // Returns count
169         } else {
170             throw new IDNotRecognisedException("Race ID not found"); // Throws a new IDNotRecognised
                    exception
171         }
172     }
173
174
175     /**
176      * @param raceId
177      * @param stageName
178      * @param description
179      * @param length
180      * @param startTime
181      * @param type
182      * @return int
183      * @throws IDNotRecognisedException
184      * @throws IllegalNameException
185      * @throws InvalidNameException
186      * @throws InvalidLengthException
187      */
188     @Override
189     public int addStageToRace(int raceId, String stageName, String description, double length,
            LocalDateTime startTime,
190             StageType type) throws IDNotRecognisedException, IllegalNameException, InvalidNameException,
                    InvalidLengthException {
191         assert raceId > 0 : "Race ID must be greater than 0"; // Throws an assertion exception if the raceId
                is less than 0
```

```
192         boolean found = false;
193         for (Race race :Races) { //For each race in races
194             if (race.getId() == raceId) { //Checks if the current race's id equals the provided raceId
195                 found = true; // Sets the found variable to true
196             }
197         }
198         if (found == false) { // Checks if the found variable is equal to false
199             throw new IDNotRecognisedException("ID of the race not found"); // Throws a new IDNotRecognised
                    exception
200         }
201         for (Stage stage : Stages) { //For each stage in Stages
202             if (stage.getStageName().equals(stageName)) { //Checks if the current stage's name equal to the
                    provided stage name
203                 throw new IllegalNameException("Stage with name already exists"); // Throws a new
                        IllegalNameException with the message "Stage with name already exists"
204             }
205         }
206         if (stageName == null) { // Checks if the stageName is equal to null
207             throw new InvalidNameException("Name cannot be Null"); // Throws a new InvalidNameException with
                    the message "Name cannot be Null"
208         }
209         if (stageName.length() > 30) { // Checks if the stageName is less than 30 characters long
210             throw new InvalidNameException("Name must be less than 30 characters"); // Throws a new
                    InvalidNameException with the message "Name must be less than 30 characters"
211         }
212         if (stageName.length() == 0){ // Checks if the length of stageName is equal to 0 character
213             throw new InvalidNameException("Name cannot be empty"); // Throws a new InvalidNameException
                    with the message "Name cannot be empty"
214         }
215         if (length < 5) { // Checks if the legnth of the stage is less than 5km
216             throw new InvalidLengthException("Stage length must be at least 5km"); // Throws a new
                    InvalidLengthException with the message "Stage length must be at least 5km"
217         }
218         Stages.add(new Stage(raceId, stageName, description, length, startTime, type)); //Creates a new
                stage and adds it to the stage list
219
220         return Stages.get(Stages.size()-1).getStageId(); // Returns the ID of the stage at the end of the
                stages list
221     }
222
223
224     /**
225      * @param raceId
226      * @return int[]
227      * @throws IDNotRecognisedException
228      */
229     @Override
230     public int[] getRaceStages(int raceId) throws IDNotRecognisedException {
231         assert raceId > 0 : "Race ID must be greater than 0"; // Throws an assertion exception if the raceId
                is less than 0
232         ArrayList<Stage> stageThings = new ArrayList<>(); // Creates a new ArrayList object called segmentIDs
233         boolean found = false;
234         for (Race Race : Races) { // For each Race in Races
235             if (Race.getId() == raceId) { // If the selected race's id is equal to the specified race id
236                 found = true; // Sets the found variable to true
```

```
237                 break; // Breaks out of the for loop ound completed
238             }
239         }
240         if (!found) { // If not found
241             throw new IDNotRecognisedException("Stage ID not recognised"); // Throws an IDNotRecognised
                    Exception
242         }
243         for (Stage stage: Stages) { // For stage, stage in the stages arrayList
244             if (stage.getRaceID() == raceId) { // Checks if the raceId for that race is equal to the stageId
245                 stageThings.add(stage); // Adds the stage to the stageThings arrayList
246             }
247         }
248         Collections.sort(stageThings); // Sorts stageThings on start time
249         int[] idArray = new int[stageThings.size()]; // Creates a new array called idArray
250         for (int i = 0; i < stageThings.size(); i++) { //For each stageID in the ArrayLiist
251            idArray[i] = stageThings.get(i).getStageId(); // Add each stageId in stageThings to idArray array
252         }
253         return idArray; // Returns the stageId list
254     }
255
256
257     /**
258      * @param stageId
259      * @return double
260      * @throws IDNotRecognisedException
261      */
262     @Override
263     public double getStageLength(int stageId) throws IDNotRecognisedException {
264         assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the
                stageId is less than 0
265         for (Stage stage : Stages) { // For each stage in Stages ArrayList
266             if (stage.getStageId() == stageId) { // Checks if the stageId for that stage is equal to the
                    stage ID provided
267                 return stage.getLength(); // Returns the length of the stage in KM
268             }
269         }
270         throw new IDNotRecognisedException("Stage ID doesn't belong to a stage"); // Throws a new
                IDNotRecognised Exception
271     }
272
273
274     /**
275      * @param stageId
276      * @throws IDNotRecognisedException
277      */
278     @Override
279     public void removeStageById(int stageId) throws IDNotRecognisedException {
280         assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the
                stageId is less than 0
281         boolean found = false; // Declares a variable found to be equal to false
282         for(Stage stage : Stages){ // Loops through the stages in the stages array list
283             if (stage.getStageId() == stageId){ // Checks if the stageId of that stage is equal to the given
                    stage id
284                 found = true; // Sets the found variable to true
285                 for (int i=0;i<Segments.size();i++){ // Loops through the loop i number of times where i is
```

```
                    equal to the legnth of the segment array list
286                 if(Segments.get(i).getStageId() == stageId){ // checks if the segment at that index in
                        the array list has the sae stage id as the given stage id
287                     Segments.remove(i); // Removes the segment at that index of the segment array list
288                 }
289             }
290             break; // Breaks from the for loop
291         }
292     }
293     if (!found){ // Checks if the found variable is equal to false
294         throw new IDNotRecognisedException("That is not a valid stage ID"); // Throws a new
                IDNotRecognised exception with the message That is not a valid team ID
295     }
296     for (int i = 0; i< Stages.size(); i++){ // Loops through the stage array list the number of times of
            the length of the list
297         if(Stages.get(i).getStageId() == stageId){ // Checks if the stage ID for that stage in the array
                list is equal to the given stage ID
298             Stages.remove(i); // Removes the stage from the array list
299             break; // Breaks from the for loop
300         }
301     }
302 }
303
304
305 /**
306  * @param stageId
307  * @param location
308  * @param type
309  * @param averageGradient
310  * @param length
311  * @return int
312  * @throws IDNotRecognisedException
313  * @throws InvalidLocationException
314  * @throws InvalidStageStateException
315  * @throws InvalidStageTypeException
316  */
317 @Override
318 public int addCategorizedClimbToStage(int stageId, Double location, SegmentType type, Double
        averageGradient,
319         Double length) throws IDNotRecognisedException, InvalidLocationException,
                InvalidStageStateException, InvalidStageTypeException {
320     assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the
            stageId is less than 0
321     boolean found = false;
322     for (Stage stage: Stages) { // Starts a for loop to loop through the stages in the stage array list
323         if (stage.getStageId() == stageId) { // Checks if the stage ID of the stage is equal to the
                given stage ID
324             found = true; // Sets the found variable to be true
325             if (stage.getLength() < location) { // If stage length is less than the distance to the end
                    of the segment
326                 throw new InvalidLocationException("The location must be less than the length of the
                        stage"); // Throws a new InvalidLocationException witth the message: The location
                        must be less than the length of the stage
327             }
328             if (stage.getType() == StageType.TT){ // If stage type is TT from enum StageType
```

```
                throw new InvalidStageTypeException("Time-trial stages cannot contain any segment"); // Throws a new InvalidStageType Exception with the message: Time-trial stages cannot contain any segment
            }
            if (stage.getWaitingForResults()) { // If stage state is waiting for results
                throw new InvalidStageStateException("Stage: " + stage.getStageName() + " is waiting for a result"); // Throws a new InvaldStageState exception with the message: "Stage: " + stage.getStageName() + " is waiting for a result"
            }


            break; // Breaks from the for loop
        }
    }
    if (!found) { // If stage with given stageId is not found
        throw new IDNotRecognisedException("Stage ID doesn't match any stages"); // Throws a new IDNotRecognisedException with the message: Stage ID doesn't match any stages

    }
    Segments.add(new Segment(stageId, location, type, averageGradient, length)); //Creates a new segment and adds it to the segment list

    return Segments.get(Segments.size()-1).getSegmentId(); // Returns the ID of the segment created
}


/**
 * @param stageId
 * @param location
 * @return int
 * @throws IDNotRecognisedException
 * @throws InvalidLocationException
 * @throws InvalidStageStateException
 * @throws InvalidStageTypeException
 */
@Override
public int addIntermediateSprintToStage(int stageId, double location) throws IDNotRecognisedException, InvalidLocationException, InvalidStageStateException, InvalidStageTypeException{
    assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the stageId is less than 0
    boolean found = false;
    for (Stage stage: Stages) { // Starts a for loop to loop through the stages in the stage array list
        if (stage.getStageId() == stageId) { // Checks if the stage ID of the stage is equal to the given stage ID
            found = true; // Sets the found variable to be true
            if (stage.getLength() < location) { // If stage length is less than the distance to the end of the segment
                throw new InvalidLocationException("The location must be less than the length of the stage"); // Throws a new InvalidLocationException witth the message: The location must be less than the length of the stage
            }
            if (stage.getType() == StageType.TT){ // If stage type is TT from enum StageType
                throw new InvalidStageTypeException("Time-trial stages cannot contain any segment"); // Throws a new InvalidStageType Exception with the message: Time-trial stages cannot contain any segment
```

```
            }
            if (stage.getWaitingForResults()) { // If stage state is waiting for results
                throw new InvalidStageStateException("Stage: " + stage.getStageName() + " is waiting for
                    a result"); // Throws a new InvaldStageState exception with the message: "Stage: " +
                    stage.getStageName() + " is waiting for a result"
            }
            break; // Breaks from the for loop
        }
    }
    if (!found) { // If stage with given stageId is not found
        throw new IDNotRecognisedException("Stage ID doesn't match any stages"); // Throws a new
            IDNotRecognisedException with the message: Stage ID doesn't match any stages

    }

    Segments.add(new Segment(stageId, location, SegmentType.SPRINT)); //Creates a new segment and adds
        it to the segment list

    return Segments.get(Segments.size()-1).getSegmentId(); // Returns the ID of the segment
}


/**
 * @param segmentId
 * @throws IDNotRecognisedException
 * @throws InvalidStageStateException
 */
@Override
public void removeSegment(int segmentId) throws IDNotRecognisedException, InvalidStageStateException {
    assert segmentId > 0 : "Segment ID must be greater than 0"; // Throws an assertion exception if the
        segmentId is less than 0
    boolean found = false; // Declares a variable called found and sets it to false
    int theStageId = 0; // Declares a new variable called theStageId
    for (Segment segment : Segments){ // For each segment in the segment array list
        if (segment.getSegmentId() == segmentId){ // Checks if the segment id for that segment is equal
            to the given segment id
            found = true; // Sets the found variable to true
            theStageId = segment.getStageId(); // Gets the stage if for the segment
        }
    }
    if (!found){ // If found is set to false
        throw new IDNotRecognisedException("Segment with that Id not found"); // throws a new
            IDNotRecognisedException with the message "Segment with that Id not found"
    }
    for (Stage stage : Stages){ // for each stage in the stages array list
        if (stage.getStageId() == theStageId){ // Checks if the stage id for that stage is equal to the
            theStageId variable
            if (stage.getWaitingForResults()){ // Checks if the stage is waiting for a reult
                throw new InvalidStageStateException("Stage is waiting for results"); // Throws a new
                    InvalidStageStateException with the message "Stage is waiting for results"
            }
        }
    }
    for (int i=0; i<Segments.size(); i++){ // Loops through the following code i number of times where i
        is the length of the segment array list
```

```
415             if (Segments.get(i).getSegmentId() == segmentId) { // Checks if the segment id for that segment
                    is equal to the given segment id
416                 Segments.remove(i); // Removes the segment from the segment array list
417                 break; // Breaks from the for loop
418             }
419         }
420 
421     }
422 
423 
424     /**
425      * @param stageId
426      * @throws IDNotRecognisedException
427      * @throws InvalidStageStateException
428      */
429     @Override
430     public void concludeStagePreparation(int stageId) throws IDNotRecognisedException,
            InvalidStageStateException {
431         assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the
                stageId is less than 0
432         boolean found = false;
433         for (Stage stage: Stages) { // For each stage in Stages
434             if (stage.getStageId() == stageId) { // If curren stage's ID is equal to providedd stageID
435                 if (stage.getWaitingForResults()) { // If current stage is waiting for results
436                     throw new InvalidStageStateException("Stage is already waiting for results"); // Throws a
                            new INvalidStageStateException with the message Stage is already waiting for result
437                 }
438                 stage.setWaitingForResults(true); // Sets current stage's WaitingForResults variable to true
439                 found = true; // Sets the found variable to true
440             }
441         }
442         if (!found) { // Checks if the found variable is set to false
443             throw new IDNotRecognisedException("Stage ID is not recognized"); // Throws a new
                    IDNotRecognised exception with the message Stage ID is not recognized
444         }
445     }
446 
447 
448     /**
449      * @param stageId
450      * @return int[]
451      * @throws IDNotRecognisedException
452      */
453     @Override
454     public int[] getStageSegments(int stageId) throws IDNotRecognisedException {
455         assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the
                stageId is less than 0
456         ArrayList<Segment> segmentThings = new ArrayList<>(); // Creates a new ArrayList object called
                segmentIDs
457         boolean found = false;
458         for (Stage stage : Stages) { // For each Stage in Stages
459             if (stage.getStageId() == stageId) { // If the selected stage's id is equal to the specified
                    stage id
460                 found = true; // Sets the found variable to true
461                 break; // Breaks out of the for loop ound completed
```

```
462             }
463         }
464         if (!found) { // If the found variable is set to false
465             throw new IDNotRecognisedException("Stage ID not recognised"); // Throws an IDNotRecognised
                    Exception
466         }
467         for (Segment segment: Segments) { // For each segment in the Segments arrayList
468             if (segment.getStageId() == stageId) { //Checks if the segmentId for that segment is equal to
                        the stageId
469                 segmentThings.add(segment); // Adds the segmentId to the segmentIds ArrayList
470             }
471         }
472         Collections.sort(segmentThings); // Sorts the arrayList on segment location
473         int[] idArray = new int[segmentThings.size()]; // Creates a new array called idArray
474         for (int i = 0; i < segmentThings.size(); i++) { //For each segment in the ArrayList
475            idArray[i] = segmentThings.get(i).getSegmentId(); // Add each segmentId in segmentThings to
                    idArray array
476         }
477         return idArray; // Returns the segmentId list
478     }
479
480
481     /**
482      * @param name
483      * @param description
484      * @return int
485      * @throws InvalidNameException
486      * @throws IllegalNameException
487      */
488     @Override
489     public int createTeam(String name, String description) throws InvalidNameException,
            IllegalNameException {
490
491         if (name == null) { // Checks if name is null
492             throw new InvalidNameException("Name cannot be Null"); // Throws a new InvalidNameException with
                    the message "Name cannot be Null"
493         }
494         if (name.length() > 30) { // Checks if the length of the name is greater than 30
495             throw new InvalidNameException("Name must be less than 30 characters"); // Throws a new
                    InvalidNameException with the message "Name must be less than 30 characters"
496         }
497         if (name.contains(" ")){ //Checks if any white space is present in the provided name
498             throw new InvalidNameException("Name must not contain white space"); // Throws a new
                    InvalidNameException with the message "Name must not contain white space"
499         }
500         if (name.equals("")){
501             throw new InvalidNameException("Name cannot be empty"); // Throws a new InvalidNameException
                    with the message "Name cannot be empty"
502         }
503         // If the name already exists in the platform. Throw IllegalNameException
504         for (Team team: Teams) { // For each race in races
505             if (team.getName().equals(name)) { // Checks if the name of the race is equal to the name
                        provided
506                 throw new IllegalNameException("Team name already exists"); // Throws a new
                        IllegalNameException with the message "Team name already exists"
```

```
507             }
508         }
509 
510         Teams.add(new Team(name, description)); //Creates a new team and adds it to the team list
511         return Teams.get(Teams.size()-1).getTeamId(); // Returns the ID of the Team at the end of the teams
                list;
512     }
513 
514 
515     /**
516      * @param teamId
517      * @throws IDNotRecognisedException
518      */
519     @Override
520     public void removeTeam(int teamId) throws IDNotRecognisedException {
521         assert teamId > 0 : "Team ID must be greater than 0"; // Throws an assertion exception if the teamId
                is less than 0
522         boolean found = false; // Declares a variable found to be equal to false
523         for(Team team : Teams){ // Loops through the teams in the teams array list
524             if (team.getTeamId() == teamId){ // Checks if the teamId of that team is equal to the given team
                    Id
525                 found = true; // Sets the found variable to true
526                 for (Rider rider : Riders){ // Loops through the riders in the riders array list
527                     if (rider.getTeamId() == teamId){ // Checks if the team id for that rider is equal to the
                            given team id
528                         removeRider(rider.getId()); // Removes the rider from the syste
529                     }
530                 }
531             }
532         }
533         if (!found){ // Checks if the found variable is equal to false
534             throw new IDNotRecognisedException("That is not a valid team ID"); // Throws a new
                    IDNotRecognised exception with the message That is not a valid team ID
535         }
536         for (int i = 0; i< Teams.size(); i++){ // Loops through the teams array list the number of times of
                the length of the list
537             if(Teams.get(i).getTeamId() == teamId){ // Checks if the team ID for that team in the array list
                    is equal to the given team ID
538                 Teams.remove(i); // Removes the team from the array list
539                 break; // Breaks from the for loop
540             }
541         }
542 
543     }
544 
545 
546     /**
547      * @return int[]
548      */
549     @Override
550     public int[] getTeams() {
551         ArrayList<Integer> TeamIDs = new ArrayList<>();
552         for (Team team : Teams) {
553             TeamIDs.add(team.getTeamId());
554         }
```

```
555         int[] idArray = new int[TeamIDs.size()];
556         for (int i = 0; i < TeamIDs.size(); i++) { //For each teamID in the ArrayLiist
557             idArray[i] = TeamIDs.get(i); // Add each value in teamIDs to idArray array
558         }
559         return idArray;
560     }
561
562
563     /**
564      * @param teamId
565      * @return int[]
566      * @throws IDNotRecognisedException
567      */
568     @Override
569     public int[] getTeamRiders(int teamId) throws IDNotRecognisedException{
570         assert teamId > 0 : "Team ID must be greater than 0"; // Throws an assertion exception if the teamId
                 is less than 0
571         ArrayList<Integer> riderIDs = new ArrayList<>(); // Creates a new ArrayList object called riderIDs
572         boolean found = false;
573         for (Team team : Teams) { // For each team in Teams
574             if (team.getTeamId() == teamId) { // If the selected riders id is equal to the specified rider id
575                 found = true; // Sets the found variable to true
576                 break; // Breaks out of the for loop ound completed
577             }
578         }
579         if (!found) {
580             throw new IDNotRecognisedException("Team ID not recognised"); // Throws an IDNotRecognised
                     Exception
581         }
582         for (Rider rider: Riders) {
583             if (rider.getTeamId() == teamId) { //Checks if the riderId for that rider is equal to the riderId
584                 riderIDs.add(rider.getId()); // Adds the riderId to the riderIds ArrayList
585             }
586         }
587         int[] idArray = new int[riderIDs.size()]; // Creates a new array called idArray
588         for (int i = 0; i < riderIDs.size(); i++) { //For each riderID in the ArrayLiist
589             idArray[i] = riderIDs.get(i); // Add each value in riderIDs to idArray array
590         }
591         return idArray; // Returns the riderId list
592     }
593
594
595     /**
596      * @param teamId
597      * @param name
598      * @param yearOfBirth
599      * @return int
600      * @throws IDNotRecognisedException
601      * @throws IllegalArgumentException
602      */
603     @Override
604     public int createRider(int teamId, String name, int yearOfBirth) throws IDNotRecognisedException,
             IllegalArgumentException {
605         assert teamId > 0 : "Team ID must be greater than 0"; // Throws an assertion exception if the teamId
                 is less than 0
```

```
606         boolean found = false;
607         if (name == null) { // Checks if name is null
608             throw new IllegalArgumentException("Name cannot be Null"); // Throws a new InvalidNameException
                    with the message "Name cannot be Null"
609         }
610         if (yearOfBirth<1900) {
611             throw new IllegalArgumentException("Year of birth must be greater than 1900"); // Throws a new
                    InvalidNameException with the message "Year of birth mus be greater than 1900"
612         }
613         // If the name already exists in the platform. Throw IllegalNameException
614         for (Team team: Teams) { // For each race in races
615             if (team.getTeamId() == (teamId)) { // Checks if the name of the race is equal to the name
                    provided
616                 found = true;
617             }
618         }
619         if (!found) {
620             throw new IDNotRecognisedException("No team with that ID was found"); // Throws a new
                    InvalidNameException with the message No team with that ID was found
621         }
622         Riders.add(new Rider(teamId, name, yearOfBirth)); //Creates a new rider and adds it to the riders
                list
623         return Riders.get(Riders.size()-1).getId(); // Returns the ID of the Rider at the end of the rider
                list;
624     }
625
626
627     /**
628      * @param riderId
629      * @throws IDNotRecognisedException
630      */
631     @Override
632     public void removeRider(int riderId) throws IDNotRecognisedException {
633         assert riderId > 0 : "Rider ID must be greater than 0"; // Throws an assertion exception if the
                riderId is less than 0
634         boolean found = false; // Creates a variable called found and sets it to false
635         for (Rider rider : Riders){ // Loops through the riders in the riders arrayList
636             if (rider.getId() == riderId){ // Checks if the riderId is equal to the Id of the rider in the
                    loop
637                 found = true; // Sets the found variable to true
638                 for (Stage stage : Stages){ // For each stage in the stage array list
639                     stage.removeRiderResults(riderId); // Removes the riders results from the hash map in
                            that stage
640                 }
641                 break;
642             }
643         }
644         if (!found){ // Checks if the found variable is equal to false
645             throw new IDNotRecognisedException("That is not a valid rider ID"); // Throws a new
                    IDNotRecognised exception with the message That is not a valid rider ID
646         }
647         for (int i = 0; i< Riders.size(); i++){ // Loops through the riders array list the number of times
                of the length of the list
648             if(Riders.get(i).getId() == riderId){ // Checks if the rider ID for that rider in the array list
                    is equal to the given rider ID
```

```
649                 Riders.remove(i); // Remoes the rider from the array list
650                 break; // Breaks from the for loop
651             }
652         }
653     }
654 
655 
656     /**
657      * @param stageId
658      * @param riderId
659      * @param checkpoints
660      * @throws IDNotRecognisedException
661      * @throws DuplicatedResultException
662      * @throws InvalidCheckpointsException
663      * @throws InvalidStageStateException
664      */
665     @Override
666     public void registerRiderResultsInStage(int stageId, int riderId, LocalTime... checkpoints) throws
          IDNotRecognisedException, DuplicatedResultException, InvalidCheckpointsException,
          InvalidStageStateException {
667         assert riderId > 0 : "Rider ID must be greater than 0"; // Throws an assertion exception if the
              riderId is less than 0
668         assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the
              stageId is less than 0
669         boolean found = false; // Creates a variable called found and sets it to false
670         for (Rider rider : Riders) { // Loops through the Riders ArrayList
671             if (rider.getId() == riderId){ // Checks if the riderId is equal to the Id of the rider in the
                  loop
672                 found = true; // Sets the found variable to true
673             }
674         }
675         if (!found) { // Checks if the found variable is set to false
676             throw new IDNotRecognisedException("That is not a valid rider ID"); // Throws a new
                  IDNotRecognised exception with the message That is not a valid rider ID
677         }
678         found = false; // Sets found to falsee
679         for (Stage stage: Stages) { // Loops through the stage ArrayList
680             if (stage.getStageId() == stageId) { //Checks if the stageId is equal to the Id of the stage in
                  the loop
681                 found = true; // Sets found to true
682                 if (stage.getStageTimes(riderId)!=null) { // Checks if the stage times array at that riders
                      Id is null
683                     throw new DuplicatedResultException("Rider already has a result"); // Throws a new
                          DuplicateResult exception with the message Rider already has a result
684                 }
685                 if (checkpoints.length != getStageSegments(stageId).length+2) { // Checks if the length of
                      the input checkpoints is not equal to the number of segments in the stage + 2
686                     throw new InvalidCheckpointsException("Incorrect number of times for that stage"); //
                          Throws a new InvalidCheckpoints exception with the message Incorrect number of times
                          for that stage
687                 }
688                 if (!stage.getWaitingForResults()){ // Checks if the stage is waiting for results
689                     throw new InvalidStageStateException("Stage is not waiting for results"); // Throws a new
                          InvalidStageState exception with the message Stage is not waiting for results
690                 }
```

```
                stage.setStageTimes(riderId, checkpoints); // Sets the stage times for the given rider
            }
        }

        if (!found) { // Checks if the found variable is equal to false
            throw new IDNotRecognisedException("That is not a valid stage ID"); // Throws a new
                IDNotRecognised exception with the message That is not a valid stage ID
        }
    }


    /**
     * @param stageId
     * @param riderId
     * @return LocalTime[]
     * @throws IDNotRecognisedException
     */
    @Override
    public LocalTime[] getRiderResultsInStage(int stageId, int riderId) throws IDNotRecognisedException {
        assert riderId > 0 : "Rider ID must be greater than 0"; // Throws an assertion exception if the
            riderId is less than 0
        assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the
            stageId is less than 0
        boolean found = false; // Creates a variable called found and sets it to false
        for (Rider rider : Riders){ // Loops through the riders in the riders arrayList
            if (rider.getId() == riderId){ // Checks if the riderId is equal to the Id of the rider in the
                loop
                found = true; // Sets the found variable to true
            }
        }
        if (!found){ // Checks if the found variable is equal to false
            throw new IDNotRecognisedException("That is not a valid rider ID"); // Throws a new
                IDNotRecognised exception with the message That is not a valid rider ID
        }
        found = false; // Sets the found variable to false
        for (Stage stage : Stages) { // For each Stage in Stages
            if (stage.getStageId() == stageId) { // If selected stageId equals the given stage id
                LocalTime[] times = stage.getStageTimes(riderId); // Times array is given the array from
                    getStageTimes
                if (times == null){ // Checks if the times variable is equal to null
                    return new LocalTime[0]; // Returns a new LocalTime array of length 0
                }
                LocalTime[] newTimes = new LocalTime[times.length]; // Creates a new array with the length of
                    the times array
                for (int i = 0; i<times.length-1; i++){ // Loops for the legnth of the times array
                    newTimes[i] = times[i+1]; // Sets the newTimes array at possition i to the value held at
                        the times array at position i + 1
                }
                newTimes[times.length-1] = times[times.length-1].minusNanos(times[0].toNanoOfDay()); //
                    Appends the LocalTime start value minus the Localtime finish value to give the elapsed
                    time at the end of the array
                return newTimes; // Returns the newTimes array
            }
        }
        if (!found){ // Checks if the found variable is equal to false
```

```
736             throw new IDNotRecognisedException("That is not a valid stage ID"); // Throws a new
                    IDNotRecognised exception with the message That is not a valid stage ID
737         }
738         return null; // Returns null
739     }
740
741
742     /**
743      * @param stageId
744      * @param riderId
745      * @return LocalTime
746      * @throws IDNotRecognisedException
747      */
748     @Override
749     public LocalTime getRiderAdjustedElapsedTimeInStage(int stageId, int riderId) throws
            IDNotRecognisedException {
750         assert riderId > 0 : "Rider ID must be greater than 0"; // Throws an assertion exception if the
                riderId is less than 0
751         assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the
                stageId is less than 0
752         boolean found = false; // Creates a variable called found and sets it to false
753         for (Rider rider : Riders){ // Loops through the riders in the riders arrayList
754             if (rider.getId() == riderId){ // Checks if the riderId is equal to the Id of the rider in the
                    loop
755                 found = true; // Sets the found variable to true
756                 break;
757             }
758         }
759         if (!found){ // Checks if the found variable is equal to false
760             throw new IDNotRecognisedException("That is not a valid rider ID"); // Throws a new
                    IDNotRecognised exception with the message That is not a valid rider ID
761         }
762         found = false; // Sets the found variable to false
763         for (Stage stage : Stages){ // Loops through the stages in the stages arraylist
764             if (stage.getStageId() == stageId){ // Checks if the stageId is equal to the stageId given
765                 found = true; // Sets the found variable to true
766                 stage.sortHashMap(); // Sorts the stage hash map
767                 LocalTime[] timeArray = getRiderResultsInStage(stageId, riderId); // Gets the riders results
                        for the given stage and puts them into tthe time array
768                 LocalTime elapsedTime = timeArray[timeArray.length-1]; // Gets the elapsed time which is
                        stored in the last index of the array
769                 int rankIncr = 1; // Sets the rankIncrement variable to 1
770                 int rank = stage.getRank(riderId); // Gets that riders rank in the stage
771                 if (rank != 1){ // Checks if the riders rank is not equal to 1
772                     while(rank > rankIncr && elapsedTime.minusNanos(((getRiderResultsInStage(stageId,
                            stage.getRiderIdFromRank(rank-rankIncr)))[timeArray.length-1]).toNanoOfDay()).toNanoOfDay()
                            < 1000000000) { // While the rank of the rider is greater than the rank increment and
                            the difference between the elapsed time of the current and next rider is less than 1
                            second
773                         elapsedTime = getRiderResultsInStage(stageId,
                                stage.getRiderIdFromRank(rank-rankIncr))[timeArray.length-1]; // Sets the elapsed
                                time to the elapsed time of that rider
774                         rankIncr++; // increments the rankIncr by one
775                     }
776
```

```
777                 }
778                 return elapsedTime; // Returns the elapsed time
779 
780             }
781         }
782         if (!found){ // Checks if the found variable is equal to false
783             throw new IDNotRecognisedException("That is not a valid stage ID"); // Throws a new
                    IDNotRecognised exception with the message That is not a valid stage ID
784         }
785         return null; // Returns null
786     }
787 
788 
789     /**
790      * @param stageId
791      * @param riderId
792      * @throws IDNotRecognisedException
793      */
794     @Override
795     public void deleteRiderResultsInStage(int stageId, int riderId) throws IDNotRecognisedException {
796         assert riderId > 0 : "Rider ID must be greater than 0"; // Throws an assertion exception if the
                riderId is less than 0
797         assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the
                stageId is less than 0
798         boolean found = false; // Creates a variable called found and sets it to false
799         for (Rider rider : Riders){ // Loops through the riders in the riders arrayList
800             if (rider.getId() == riderId){ // Checks if the riderId is equal to the Id of the rider in the
                    loop
801                 found = true; // Sets the found variable to true
802                 break;
803             }
804         }
805         if (!found){ // Checks if the found variable is equal to false
806             throw new IDNotRecognisedException("That is not a valid rider ID"); // Throws a new
                    IDNotRecognised exception with the message That is not a valid rider ID
807         }
808         found = false; // Sets the variable found to false
809         for (Stage stage : Stages){ // loops through all the stages in the stages array list
810             if (stage.getStageId() == stageId){ // Checks if the stage ID is equal to the given stage ID
811                 found = true; // Sets the found variable to true
812                 stage.removeRiderResults(riderId); // Removes the riders results from the hashmap
813             }
814         }
815         if (!found){ // Checks if the found variable is equal to false
816             throw new IDNotRecognisedException("That is not a valid stage ID"); // Throws a new
                    IDNotRecognised exception with the message That is not a valid stage ID
817         }
818 
819 
820     }
821 
822 
823     /**
824      * @param stageId
825      * @return int[]
```

```
826         * @throws IDNotRecognisedException
827         */
828        @Override
829        public int[] getRidersRankInStage(int stageId) throws IDNotRecognisedException {
830            assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the
                stageId is less than 0
831            boolean found = false; // Creates a found variable and sets it to false
832            int count = 0; // Creates a count variable and sets it to 0
833            for (Stage stage : Stages){ // Loops through the stages in the stages array list
834                if (stage.getStageId() == stageId){ // Checks if the stage ID is equal to the given stageId
835                    found = true; // Sets the found variable to true
836                    Stages.get(count).sortHashMap(); // Sorts the hashmap in the stage
837                    HashMap<Integer, LocalTime[]> map = stage.getHashMap(); // Gets the hashmap from the stage
                        and sets the map variable to it
838                    int[] riderIds = new int[map.size()]; // Creates a riderIds array with the size of the map
839                    int count2 = 0; // Creates a count2 variable and sets it to 0
840                    for (int key : map.keySet()){ // Loops through key key in the key set of the map
841                        riderIds[count2] = key; // Inserts the key into the riderIds array at the index of count2
842                        count2++; // Increments the count2 variable by 1
843                    }
844                    return riderIds; // Returns the riderIds
845                }
846                count++; // Increments the count variable by 1
847            }
848            if (!found){ // Checks if the found variable is equal to false
849                throw new IDNotRecognisedException("StageId not recognised"); // Throws a new
                    IDNotRecognisedException with the message "StageId not recognised"
850            }
851
852            return null; // returns null
853        }
854
855
856        /**
857         * @param stageId
858         * @return LocalTime[]
859         * @throws IDNotRecognisedException
860         */
861        @Override
862        public LocalTime[] getRankedAdjustedElapsedTimesInStage(int stageId) throws IDNotRecognisedException {
863            assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the
                stageId is less than 0
864            boolean found = false; // creates a found variable and sets it to false
865            int count = 0; // Creates a count variable and sets it to 0
866            for (Stage stage : Stages){ // Loops through the stages in the stages array list
867                if (stage.getStageId() == stageId){ // Checks if the stage Id of that stage is equal to the
                    given stageID
868                    found = true; // Sets the found variable to true
869                    Stages.get(count).sortHashMap(); // Sorts the results hashmap of that stage
870                    HashMap<Integer, LocalTime[]> map = stage.getHashMap(); // Creates a new hashmap called map
                        and sets it equal to the sorted hashmap of the stage
871                    LocalTime[] finishTimes = new LocalTime[map.size()]; // Creates a new localtime array with
                        the size of the map hashmap
872                    int count2 = 0; // Creates a count2 variable and sets it equal to 0
873                    for (LocalTime[] value : map.values()){ // loops through the values of the map values
```

```
874                 finishTimes[count2] = value[value.length-1]; // Sets the finish times array at the index
                        of count2 to the value at the last index of the value variable
875                 count2++; // Adds one to the count2 variable
876             }
877             return finishTimes; // Returns the finishTimes array
878         }
879         count++; // Adds one to the count variable
880     }
881     if (!found){ // Checks if the found variable is equal to false
882         throw new IDNotRecognisedException("StageId not recognised"); // Throews a new
                IDNotRecognisedException with the message "StageId not recognised"
883     }
884     return null; // Return null
885 }
886 
887 
888 /**
889  * @param stageId
890  * @return int[]
891  * @throws IDNotRecognisedException
892  */
893 @Override
894 public int[] getRidersPointsInStage(int stageId) throws IDNotRecognisedException {
895     assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the
            stageId is less than 0
896     boolean found = false; // Creates a found variable and sets it equal to false
897     StageType type = null; // Creates a type variable with type StageType and sets it equal to null
898     HashMap<Integer, LocalTime[]> timesMap = new HashMap<Integer,LocalTime[]>(); // Creates a new
            hashmap called timesMap
899     LinkedHashMap <Integer, Integer> pointsMap = new LinkedHashMap<Integer,Integer>(); // Creates a new
            Linkedhashmap called pointsMap
900 
901     for (Stage stage : Stages){ // loops through all the stages in the stages array list
902         if (stage.getStageId() == stageId){ // Checks if the stage id of that stage is equal to the
                given stageId
903             found = true; // Sets the found variable to true
904             stage.sortHashMap(); // Sorts the hash map of that stage
905             timesMap = stage.getHashMap(); // Gets the sorted hash map and sets the timeMap variable to it
906             type = stage.getType(); // Sets the type variable to the type of that stage
907             for (int key : timesMap.keySet()){ // Loops through the keys in the key set of the timesMap
908                 pointsMap.put(key,0); // Adds that key to the pointsMap
909             }
910 
911             switch (type) {
912                 case FLAT: // If type is FLAT
913                     int[] flatPoints = {50,30,20,18,16,14,12,10,8,7,6,5,4,3,2}; // Sets the flatPoints
                            variable to the points provided
914                     for (Rider rider : Riders){ // Loops through the riders in the riders array list
915                         if (pointsMap.containsKey(rider.getId()) && stage.getRank(rider.getId()) <= 15){
                                // Checks if the pointsMap contains the rider id and the stage rank of the
                                rider is less than or equal to 15
916                             pointsMap.put(rider.getId(), flatPoints[stage.getRank(rider.getId())-1]); //
                                    Adds the rider id and the riders rank to the pointsMap
917                         }
918                     }
```

```
                break; // breaks from the loop
            case TT: // If type is TT
                int[] ttPoints = {20,17,15,13,11,10,9,8,7,6,5,4,3,2,1}; // Sets the ttPoints variable
                    to the points provided
                for (Rider rider : Riders){ // Loops through the riders in the riders array list
                    if (pointsMap.containsKey(rider.getId()) && stage.getRank(rider.getId()) <= 15){
                        // Checks if the pointsMap contains the rider id and the stage rank of the
                        rider is less than or equal to 15
                        pointsMap.put(rider.getId(), ttPoints[stage.getRank(rider.getId())-1]); //
                            Adds the rider id and the riders rank to the pointsMap
                    }
                }
                break; // Breaks from the loop
            case MEDIUM_MOUNTAIN: // If the type is MEDIUM_MOUNTAIN
                int[] mmPoints = {30,25,22,19,17,15,13,11,9,7,6,5,4,3,2}; // Sets the mmPoints
                    variable to the points provided
                for (Rider rider : Riders){ // Loops through the riders in the riders array list
                    if (pointsMap.containsKey(rider.getId()) && stage.getRank(rider.getId()) <= 15){
                        // Checks if the pointsMap contains the rider id and the stage rank of the
                        rider is less than or equal to 15
                        pointsMap.put(rider.getId(), mmPoints[stage.getRank(rider.getId())-1]); //
                            Adds the rider id and the riders rank to the pointsMap
                    }
                }
                break; // Breaks from the loop
            default: // If the type is something else
                int[] hmPoints = {20,17,15,13,11,10,9,8,7,6,5,4,3,2,1}; // Sets the hmPoints variable
                    to the points provided
                for (Rider rider : Riders){ // Loops through the riders in the riders array list
                    if (pointsMap.containsKey(rider.getId()) && stage.getRank(rider.getId()) <= 15){
                        // Checks if the pointsMap contains the rider id and the stage rank of the
                        rider is less than or equal to 15
                        pointsMap.put(rider.getId(), hmPoints[stage.getRank(rider.getId())-1]); //
                            Adds the rider id and the riders rank to the pointsMap
                    }
                }
                break; // breaks from the loop
        }
        for (Segment segment : Segments){ // Loops through the segments in the segment array list
            if (stageId == segment.getStageId()) { // Checks if the stage id is equal to the given
                stage id
                SegmentType segmentType = segment.getType(); // sets the segmentType variable to the
                    type of the segment
                if (segmentType == SegmentType.SPRINT){ // Checks if the segmet type is equal to
                    SPRINT
                    for (Rider rider : Riders){ // loops through the riders in the riders array list
                        if (pointsMap.containsKey(rider.getId()) && stage.getRank(rider.getId()) <=
                            15){ // Checks if the pointsMap contains the rider id and the stage rank
                            for that rider is less than or equal to 15
                            int[] spPoints = {20,17,15,13,11,10,9,8,7,6,5,4,3,2,1}; // Sets the
                                spPoints variable to the points provided
                            pointsMap.put(rider.getId(), pointsMap.get(rider.getId()) +
                                spPoints[stage.getRank(rider.getId())-1]); // Adds the rider id and
                                riders points to the pointsMap
                        }
```

```
954                         }
955                     }
956                 }
957             }
958         }
959     }
960     if (!found) { // Checks if the found variable is false
961         throw new IDNotRecognisedException("Stage ID is not recognised"); // Throws a new
                IDNotRecognisedException with the message "Stage ID is not recognised"
962     }
963     int[] pointsArray = new int[pointsMap.size()]; // Creates a new pointsArray with the size of the
            pointsMap
964     int count = 0; // Sets the count variable to 0
965     for (int value : pointsMap.values()) { // Loops through the values in the pointsMap value set
966         pointsArray[count] = value; // Sets the value of the pointsArray at the index of count to this
                variable
967         count++; // Increments the count variable by one
968     }
969     return pointsArray; // Returns the pointsArray
970 }
971
972
973 /**
974  * @param stageId
975  * @return int[]
976  * @throws IDNotRecognisedException
977  */
978 @Override
979 public int[] getRidersMountainPointsInStage(int stageId) throws IDNotRecognisedException {
980     assert stageId > 0 : "Stage ID must be greater than 0"; // Throws an assertion exception if the
            stageId is less than 0
981     boolean found = false; // Creates a variable callled found and sets it to false
982     HashMap<Integer, LocalTime[]> timesMap = new HashMap<Integer,LocalTime[]>(); // Creates a new
            HashMap called timesMap
983     LinkedHashMap <Integer, Integer> pointsMap = new LinkedHashMap<Integer,Integer>(); // Creates a new
            LinkedHashMap called pointsMap
984     for (Stage stage : Stages){ // Loops through the stages in the stages array list
985         if (stage.getStageId() == stageId){ // Checks if the stage id of that stage is equal to the
                given stage id
986             found = true; // Sets found to true
987             stage.sortHashMap(); // Sorts the results hashmap of that stage
988             timesMap = stage.getHashMap(); // Sets the timeMap to the hashmap of that stage
989             for (int key : timesMap.keySet()){ // Loops through the keys in the timeMap key set
990                 pointsMap.put(key,0); // Adds the key to the pointsMap
991             }
992             for (Segment segment : Segments) { // Loops through the segments in the segments arrat list
993                 if (segment.getStageId() == stageId){ // Checks if the stage id of that segment is equal
                        to the given stage id
994                     if (segment.getType() != SegmentType.SPRINT){ // Checks if the segment type is SPRINT
995                         switch (segment.getType()) {
996                             case C1: // If the segment type is C1
997                                 int[] c1Points = {10,8,6,4,2,1}; // Sets the c1Points variable to the
                                        points provided
998                                 for (Rider rider : Riders){ // Loops through the riders in the riders array
                                        list
```

```
999                 if (pointsMap.containsKey(rider.getId()) &&
                        stage.getRank(rider.getId()) <= 6){ // Checks if the pointsMap
                        contains the rider id and the riders rank for that stage is less
                        than or equal to 6
1000                    pointsMap.put(rider.getId(), pointsMap.get(rider.getId()) +
                            c1Points[stage.getRank(rider.getId())-1]); // Adds the rider id
                            and the riders points to the pointsMap
1001                }
1002            }
1003            break; // Breaks from the switch case
1004        case C2:// If the segment type is C2
1005            int[] c2Points = {5,3,2,1}; // Sets the c2Points variable to the points
                    provided
1006            for (Rider rider : Riders){ // Loops through the riders in the riders array
                    list
1007                if (pointsMap.containsKey(rider.getId()) &&
                        stage.getRank(rider.getId()) <= 4){ // Checks if the pointsMap
                        contains the rider id and the riders rank for that stage is less
                        than or equal to 4
1008                    pointsMap.put(rider.getId(), pointsMap.get(rider.getId()) +
                            c2Points[stage.getRank(rider.getId())-1]); // Adds the rider id
                            and the riders points to the pointsMap
1009                }
1010            }
1011            break; // Breaks from the switch case
1012        case C3: // If the segment type is c3
1013            int[] c3Points = {2,1}; // Sets the c3Points variable to the points provided
1014            for (Rider rider : Riders){ // Loops through the riders in the riders array
                    list
1015                if (pointsMap.containsKey(rider.getId()) &&
                        stage.getRank(rider.getId()) <= 2){ // Checks if the pointsMap
                        contains the rider id and the riders rank for that stage is less
                        than or equal to 2
1016                    pointsMap.put(rider.getId(), pointsMap.get(rider.getId()) +
                            c3Points[stage.getRank(rider.getId())-1]); // Adds the rider id
                            and the riders points to the pointsMap
1017                }
1018            }
1019            break; // Breaks from the switch case
1020        case C4: // If the segment type is c4
1021            int[] c4Points = {1}; // Sets the c4Points variable to the points provided
1022            for (Rider rider : Riders){ // Loops through the riders in the riders array
                    list
1023                if (pointsMap.containsKey(rider.getId()) &&
                        stage.getRank(rider.getId()) <= 1){ // Checks if the pointsMap
                        contains the rider id and the riders rank for that stage is less
                        than or equal to 1
1024                    pointsMap.put(rider.getId(), pointsMap.get(rider.getId()) +
                            c4Points[stage.getRank(rider.getId())-1]); // Adds the rider id
                            and the riders points to the pointsMap
1025                }
1026            }
1027            break; // Breaks from the switch case
1028        default: // If the type is something else
1029            int[] hcPoints = {20,15,12,10,8,6,4,2}; // Sets the hcPoints variable to
```

```
                                    the points provided
                                for (Rider rider : Riders){ // Loops through the riders in the riders array
                                    list
                                    if (pointsMap.containsKey(rider.getId()) &&
                                        stage.getRank(rider.getId()) <= 8){ // Checks if the pointsMap
                                        contains the rider id and the riders rank for that stage is less
                                        than or equal to 8
                                        pointsMap.put(rider.getId(), pointsMap.get(rider.getId()) +
                                            hcPoints[stage.getRank(rider.getId())-1]); // Adds the rider id
                                            and the riders points to the pointsMap
                                    }
                                }
                                break; // Breaks from the switch case
                            }
                        }
                    }
                }
            }
        }
        if (!found){ // If the found variable is equal to false
            throw new IDNotRecognisedException("Stage ID is not recognised"); // Throws a new
                IDNotRecognisedException with the message "Stage ID is not recognised"
        }
        int[] pointsArray = new int[pointsMap.size()]; // Creates a new pointsArray with type int and size
             of the pointsMap
        int count = 0; // Create a variable called count with value 0
        for (int value : pointsMap.values()) { // Loops through values in the values set of the pointsMap
            pointsArray[count] = value; // Sets the value of pointsArray at the index of count to this value
            count++; // Increments the count variable by 1
        }
        return pointsArray; // Return the pointsArray
    }

    @Override
    public void eraseCyclingPortal() {
        Races.get(0).clearNumberOfRaces(); // Sets the number of races in the system to 0
        Stages.get(0).clearNumberOfStages(); // Sets the number of stages in the system to 0
        Stages.get(0).clearNumberOfStages(); // Clears stageTimes and tempStageTimes
        Segments.get(0).clearNumberOfSegments(); // Sets the number of segments in the system to 0
        Riders.get(0).clearNumberOfRiders(); // sets the number of riders in the system to 0
        Teams.get(0).clearNumberOfTeams(); // Sets the number of teams in the system to 0
        Races.clear(); // Clears the Races array list
        Stages.clear(); // Clears the Stages array list
        Segments.clear(); // Clears the Segments array list
        Riders.clear(); // Clears the Riders array list
        Teams.clear(); // Clears the Teams array list
    }


    /**
     * @param filename
     * @throws IOException
     */
    @Override
    public void saveCyclingPortal(String filename) throws IOException {
```

```
        try {
            //Creating FileOutputStream object.
            File file = new File(filename+".ser"); // Creates new File object with the name of the supplied
                filename + ".ser"
            FileOutputStream fos =
            new FileOutputStream(file);
            //Creating ObjectOutputStream object.
            ObjectOutputStream oos = new ObjectOutputStream(fos);

            //write object.
            oos.writeObject(this);

            //close streams.
            oos.close();
            fos.close();


         }catch(IOException e)
         {
             throw new IOException("There was a problem when trying to save to the file"); // Throws new
                 IOException with message "There was a problem when trying to save to the file"
         }
    }


    /**
     * @param filename
     * @throws IOException
     * @throws ClassNotFoundException
     */
    @Override
    public void loadCyclingPortal(String filename) throws IOException, ClassNotFoundException {
        try{
      CyclingPortal portal = null;
      //Creating FileOutputStream object.
      FileInputStream fis =
            new FileInputStream(filename+".ser");

            //Creating ObjectOutputStream object.
            ObjectInputStream ois = new ObjectInputStream(fis);

            //write object.
            portal = (CyclingPortal) ois.readObject();

            //close streams.
            ois.close();
            fis.close();
            this.Races = portal.Races;
            this.Stages = portal.Stages;
            this.Segments = portal.Segments;
            this.Teams = portal.Teams;
            this.Riders = portal.Riders;

            this.Races.get(0).setNumberOfRaces(Races.size());
            this.Stages.get(0).setNumberOfStages(Stages.size());
```

```
            this.Segments.get(0).setNumberOfSegments(Segments.size());
            this.Teams.get(0).setNumberOfTeams(Teams.size());
            this.Riders.get(0).setNumberOfRiders(Riders.size());

        } catch(IOException e) {
            throw new IOException("There was a problem reading that file"); // Throws new IOException
                "There was a problem reading that file");
        } catch(ClassNotFoundException e) {
            throw new ClassNotFoundException("Class files were not found"); // Throws new
                ClassNotFoundException "Class files were not found"
        }
    }



    /**
     * @param name
     * @throws NameNotRecognisedException
     */
    @Override
    public void removeRaceByName(String name) throws NameNotRecognisedException{
        boolean found = false; // Declares a variable found and sets it equal to false
        int raceId = 0;
        for (Race race : Races){ // Loops through all the races in the race array list
            if (race.getName()==name){ // Checks if the race name for that race is equal to the given race
                name
                found = true; // Sets found equal to true
                raceId = race.getId();
                for (Stage stage : Stages){ // Loops through all the stages in the stage array list
                    if (race.getId() == stage.getRaceID()){ // Checks if the Id of the race is equal to the
                        race id in the stage
                        for (int i=0;i<Segments.size();i++){ // Loops through the loop i number of times
                            where i is equal to the length of the segment array list
                            if(Segments.get(i).getStageId() == stage.getStageId()){ // checks if the segment
                                at that index in the array list has the same stage id as the current stage id
                                Segments.remove(i); // Removes the segment at that index of the segment array
                                    list
                            }
                        }
                    }
                }
                for (int i=0;i<Stages.size();i++){ // Loops through the loop i number of times where i is
                    equal to the legnth of the stages array list
                    if(Stages.get(i).getRaceID() == raceId){ // checks if the race at that index in the array
                        list has the same race id as the current race id
                        Stages.remove(i); // Removes the stage at that index of the stage array list
                    }
                }
                break; // Breaks from the for loop
            }
        }
        if (!found){ // Checks if the found variable is equal to false
            throw new NameNotRecognisedException("That Race name is not recognised"); // Throws a new
                NameNotRecognisedExcception with message: 'That Race name is not recognised'
```

```
1174        }
1175        for (int i=0; i<Races.size(); i++){ // Loops through the following code i number of times, where i
                is the length of the races array list
1176            if (Races.get(i).getName() == name){ // Checks if the race id for that race is equal to the
                    given race id
1177                Races.remove(i); // Removes that race from the system
1178                break; // Breaks from the for loop
1179            }
1180        }
1181    }
1182
1183
1184    /**
1185     * @param raceId
1186     * @return int[]
1187     * @throws IDNotRecognisedException
1188     */
1189    @Override
1190    public int[] getRidersGeneralClassificationRank(int raceId) throws IDNotRecognisedException {
1191        assert raceId > 0 : "Race ID must be greater than 0"; // Throws an assertion exception if the raceId
                is less than 0
1192        boolean found = false; // Sets variable found to false
1193        for (Race race: Races) { // For each race in Races
1194            if (race.getId() == raceId) { // If current race's id is equal too the given race id
1195                found = true; // Sets found variable to true
1196                break;
1197            }
1198        }
1199
1200        if (!found) { // If not found
1201            throw new IDNotRecognisedException("Race ID not recognized"); // Throws a new
                    IDNotRecognisedException with message "Race ID not recognized"
1202        }
1203
1204        LinkedHashMap<Integer, LocalTime> totalTimes = new LinkedHashMap<Integer,LocalTime>(); // Create new
                totalTimes LinkedHashMap
1205        LinkedHashMap<Integer, LocalTime> tempTotalTimes = new LinkedHashMap<Integer, LocalTime>(); //
                Create new tempTotalTimes LinkedHashMap
1206        for (Rider rider : Riders){ // For each rider in Riders ArrayList
1207            totalTimes.put(rider.getId(), LocalTime.of(0,0)); // Give each rider an initial time of 00:00
1208            for (Stage stage : Stages){ // For each stage in Stages ArrayList
1209                if (stage.getRaceID() == raceId){ // If current stage race id matches the given race id
1210                    LocalTime adjElapsedTime = getRiderAdjustedElapsedTimeInStage(stage.getStageId(),
                            rider.getId()); // LocalTime variable 'adjElapsedTime' is set to the elapsed time
                            returned for the current rider in the current stage
1211                    totalTimes.put(rider.getId(),
                            totalTimes.get(rider.getId()).plusNanos(adjElapsedTime.toNanoOfDay())); // Replaces
                            the current value in totalTimes for the current rider with the current time value +
                            the current stage's returned elapsed time
1212                }
1213            }
1214        }
1215        List list = new LinkedList(totalTimes.entrySet()); // Creates a new LinkedList with the values of
                totalTimes' entrySet
1216        //Custom Comparator
```

```
1217        Collections.sort(list, new Comparator() {
1218        public int compare(Object o1, Object o2) {
1219            return ((Comparable) ((LocalTime)((Map.Entry)
                (o1)).getValue())).compareTo(((LocalTime)((Map.Entry) (o2)).getValue())); // Return the
                result of comparing the 2 given times
1220            }
1221        });
1222
1223        //copying the sorted list in HashMap to preserve the iteration order
1224        HashMap sortedHashMap = new LinkedHashMap(); // Create new HashMap
1225        for (Iterator it = list.iterator(); it.hasNext();){ // For each iterator in the list
1226            Map.Entry entry = (Map.Entry) it.next(); // Sets the current iteration from the list to a
                Map.Entry variable
1227            sortedHashMap.put(entry.getKey(), entry.getValue()); // Puts the current key and value from the
                linked list into the HashMap
1228        }
1229        Map<Integer, LocalTime> map = sortedHashMap; // Copy sortedHashMap to a new Map called map
1230        Set set2 = map.entrySet(); // Puts the value of map's entry set into a Set
1231        Iterator iterator2 = set2.iterator(); // Create new iterator from the set
1232        while(iterator2.hasNext()){ // While the iterator has a next value
1233        Map.Entry me2 = (Map.Entry)iterator2.next(); // Store the next value in iterator in me2
1234        tempTotalTimes.put((Integer)me2.getKey(), (LocalTime)me2.getValue()); // Puts the key and value from
             me2 into tempTotalTimes LinkedHashMap
1235        }
1236        totalTimes = tempTotalTimes; // Sets totalTimes to store tempTotalTimes LinkedHashMap
1237        int[] toReturn = new int[totalTimes.size()]; // Create new integer array with size being the size of
             totalTimes
1238        int count = 0; // Sets integer count to 0
1239        for (int key : totalTimes.keySet()){ // For each key in totalTimes' key set
1240            toReturn[count] = key; // Puts key into the array at position count
1241            count++; // Increment count by 1
1242        }
1243        return toReturn; // Return the list
1244    }
1245
1246
1247    /**
1248     * @param raceId
1249     * @return LocalTime[]
1250     * @throws IDNotRecognisedException
1251     */
1252    @Override
1253    public LocalTime[] getGeneralClassificationTimesInRace(int raceId) throws IDNotRecognisedException {
1254        assert raceId > 0 : "Race ID must be greater than 0"; // Throws an assertion exception if the raceId
             is less than 0
1255        boolean found = false; // Sets variable found to false
1256        for (Race race: Races) { // For each race in Races
1257            if (race.getId() == raceId) { // If current race's id is equal too the given race id
1258                found = true; // Sets found variable to true
1259                break;
1260            }
1261        }
1262
1263        if (!found) { // If not found
1264            throw new IDNotRecognisedException("Race ID not recognized"); // Throws a new
```

```
            IDNotRecognisedException with message "Race ID not recognized"
1265        }
1266
1267        LinkedHashMap<Integer, LocalTime> totalTimes = new LinkedHashMap<Integer,LocalTime>(); // Create new
                totalTimes LinkedHashMap
1268        LinkedHashMap<Integer, LocalTime> tempTotalTimes = new LinkedHashMap<Integer, LocalTime>(); //
                Create new tempTotalTimes LinkedHashMap
1269        for (Rider rider : Riders){ // For each rider in Riders ArrayList
1270            totalTimes.put(rider.getId(), LocalTime.of(0,0)); // Give each rider an initial time of 00:00
1271            for (Stage stage : Stages){ // For each stage in Stages ArrayList
1272                if (stage.getRaceID() == raceId){ // If current stage race id matches the given race id
1273                    LocalTime adjElapsedTime = getRiderAdjustedElapsedTimeInStage(stage.getStageId(),
                            rider.getId()); // LocalTime variable 'adjElapsedTime' is set to the elapsed time
                            returned for the current rider in the current stage
1274                    totalTimes.put(rider.getId(),
                            totalTimes.get(rider.getId()).plusNanos(adjElapsedTime.toNanoOfDay())); // Replaces
                            the current value in totalTimes for the the current rider with the current time value
                            + the current stage's returned elapsed time
1275                }
1276            }
1277        }
1278        List list = new LinkedList(totalTimes.entrySet()); // Creates a new LinkedList with the values of
                totalTimes' entrySet
1279        //Custom Comparator
1280        Collections.sort(list, new Comparator() {
1281        public int compare(Object o1, Object o2) {
1282            return ((Comparable) ((LocalTime)((Map.Entry)
                    (o1)).getValue())).compareTo(((LocalTime)((Map.Entry) (o2)).getValue())); // Return the
                    result of comparing the 2 given times
1283            }
1284        });
1285
1286        //copying the sorted list in HashMap to preserve the iteration order
1287        HashMap sortedHashMap = new LinkedHashMap(); // Create new HashMap
1288        for (Iterator it = list.iterator(); it.hasNext();){ // For each iterator in the list
1289            Map.Entry entry = (Map.Entry) it.next(); // Sets the current iteration from the list to a
                    Map.Entry variable
1290            sortedHashMap.put(entry.getKey(), entry.getValue()); // Puts the current key and value from the
                    linked list into the HashMap
1291        }
1292        Map<Integer, LocalTime> map = sortedHashMap; // Copy sortedHashMap to a new Map called map
1293        Set set2 = map.entrySet(); // Puts the value of map's entry set into a Set
1294        Iterator iterator2 = set2.iterator(); // Create new iterator from the set
1295        while(iterator2.hasNext()){ // While the iterator has a next value
1296        Map.Entry me2 = (Map.Entry)iterator2.next(); // Store the next value in iterator in me2
1297        tempTotalTimes.put((Integer)me2.getKey(), (LocalTime)me2.getValue()); // Puts the key and value from
                me2 into tempTotalTimes LinkedHashMap
1298        }
1299        totalTimes = tempTotalTimes; // Sets totalTimes to store tempTotalTimes LinkedHashMap
1300        LocalTime[] toReturn = new LocalTime[totalTimes.size()]; // Create new LocalTime array with size
                being the size of totalTimes
1301        int count = 0; // Sets integer count to 0
1302        for (LocalTime value : totalTimes.values()){ // For each value in totalTimes' set of values
1303            toReturn[count] = value; // Puts value into the array at position count
1304            count++; // Increment count by 1
```

```
1305            }
1306            return toReturn; // Return the list
1307        }
1308
1309
1310        /**
1311         * @param raceId
1312         * @return int[]
1313         * @throws IDNotRecognisedException
1314         */
1315        @Override
1316        public int[] getRidersPointsInRace(int raceId) throws IDNotRecognisedException {
1317            assert raceId > 0 : "Race ID must be greater than 0"; // Throws an assertion exception if the raceId
                    is less than 0
1318            boolean found = false; // Sets variable found to false
1319            LinkedHashMap<Integer,Integer> adjPointsMap = new LinkedHashMap<Integer,Integer>(); // Creates new
                    LinkedHashMap called adjPointsMap
1320            for (Race race: Races) { // For each race in Races
1321                if (race.getId() == raceId) { // If current race's id is equal too the given race id
1322                    found = true; // Sets found variable to true
1323                    break;
1324                }
1325            }
1326            if (!found) { // If not found
1327                throw new IDNotRecognisedException("Race ID not recognized"); // Throws a new
                        IDNotRecognisedException with message "Race ID not recognized"
1328            }
1329            for (Rider rider : Riders){ // For each rider in Riders ArrayList
1330                adjPointsMap.put(rider.getId(), 0); // Gives each rider an initial value of 0 points
1331            }
1332            for (Stage stage : Stages){ // For each stage in Stages ArrayList
1333                if (stage.getRaceID() == raceId) { // If current stage race id matches the given race id
1334                    int[] riderIds = getRidersRankInStage(stage.getStageId()); // Creates new integer array
                            storing the ordered rider ids for the stage
1335                    int[] riderPoints = getRidersPointsInStage(stage.getStageId()); // Creates new integer array
                            storing the points in the stage, ordered corresponding to the riderIds array
1336                    for (int i=0; i<riderIds.length; i++){ // For i in range 0 - riderIds' length
1337                        adjPointsMap.put(riderIds[i], adjPointsMap.get(riderIds[i]) + riderPoints[i]); //
                                Replaces the current points in totalTimes for the current rider with their current
                                points + the current stage's returned points
1338                    }
1339                }
1340            }
1341            int[] GCRiderIds = getRidersGeneralClassificationRank(raceId); // Creates new integer array storing
                    the returned rider ids returned for the current race's general classification
1342            int[] points = new int[GCRiderIds.length]; // Creates a new integer array with a size of the length
                    of GCRiderIds (Number of riders in the race)
1343            for (int i=0;i<GCRiderIds.length; i++) { // For i in range 0 - length of GCRiderIds
1344                points[i] = adjPointsMap.get(GCRiderIds[i]); // Stores in points at position i, the value in
                        adjPointsMap corresponding to the key which is the value of GCRiderIds at position i
1345            }
1346            return points; // Returns the points array
1347        }
1348
1349
```

```
1350    /**
1351     * @param raceId
1352     * @return int[]
1353     * @throws IDNotRecognisedException
1354     */
1355    @Override
1356    public int[] getRidersMountainPointsInRace(int raceId) throws IDNotRecognisedException{
1357        assert raceId > 0 : "Race ID must be greater than 0"; // Throws an assertion exception if the raceId
                is less than 0
1358        boolean found = false; // Sets variable found to false
1359        LinkedHashMap<Integer,Integer> adjPointsMap = new LinkedHashMap<Integer,Integer>(); // Creates new
                LinkedHashMap called adjPointsMap
1360        for (Race race: Races) { // For each race in Races
1361            if (race.getId() == raceId) { // If current race's id is equal too the given race id
1362                found = true; // Sets found variable to true
1363                break;
1364            }
1365        }
1366        if (!found) { // If not found
1367            throw new IDNotRecognisedException("Race ID not recognized"); // Throws a new
                    IDNotRecognisedException with message "Race ID not recognized"
1368        }
1369        for (Rider rider : Riders){ // For each rider in Riders ArrayList
1370            adjPointsMap.put(rider.getId(), 0); // Gives each rider an initial value of 0 points
1371        }
1372        for (Stage stage : Stages){ // For each stage in Stages ArrayList
1373            if (stage.getRaceID() == raceId) { // If current stage race id matches the given race id
1374                int[] riderIds = getRidersRankInStage(stage.getStageId()); // Creates new integer array
                        storing the ordered rider ids for the stage
1375                int[] riderPoints = getRidersMountainPointsInStage(stage.getStageId()); // Creates new
                        integer array storing the mountain points in the stage, ordered corresponding to the
                        riderIds array
1376                for (int i=0; i<riderIds.length; i++){ // For i in range 0 - riderIds' length
1377                    adjPointsMap.put(riderIds[i], adjPointsMap.get(riderIds[i]) + riderPoints[i]); //
                            Replaces the current points in totalTimes for the current rider with their current
                            mountain points + the current stage's returned mountain points
1378                }
1379            }
1380        }
1381        int[] GCRiderIds = getRidersGeneralClassificationRank(raceId); // Creates new integer array storing
                the returned rider ids returned for the current race's general classification
1382        int[] points = new int[GCRiderIds.length]; // Creates a new integer array with a size of the length
                of GCRiderIds (Number of riders in the race)
1383        for (int i=0;i<GCRiderIds.length; i++) { // For i in range 0 - length of GCRiderIds
1384            points[i] = adjPointsMap.get(GCRiderIds[i]); // Stores in points at position i, the value in
                    adjPointsMap corresponding to the key which is the value of GCRiderIds at position i
1385        }
1386        return points; // Returns the points array
1387    }
1388
1389
1390    /**
1391     * @param raceId
1392     * @return int[]
1393     * @throws IDNotRecognisedException
```

```
1394      */
1395     @Override
1396     public int[] getRidersPointClassificationRank(int raceId) throws IDNotRecognisedException {
1397         assert raceId > 0 : "Race ID must be greater than 0"; // Throws an assertion exception if the raceId
              is less than 0
1398         boolean found = false; // Sets variable found to false
1399         LinkedHashMap<Integer,Integer> pointsMap = new LinkedHashMap<Integer,Integer>(); // Creates new
              LinkedHashMap called pointsMap
1400         for (Race race: Races) { // For each race in Races
1401             if (race.getId() == raceId) { // If current race's id is equal too the given race id
1402                 found = true; // Sets found variable to true
1403                 break;
1404             }
1405         }
1406         if (!found) { // If not found
1407             throw new IDNotRecognisedException("Race ID not recognized"); // Throws a new
                  IDNotRecognisedException with message "Race ID not recognized"
1408         }
1409         for (Rider rider : Riders){ // For each rider in Riders ArrayList
1410             pointsMap.put(rider.getId(), 0); // Gives each rider an initial value of 0 points
1411         }
1412         for (Stage stage : Stages){ // For each stage in Stages ArrayList
1413             if (stage.getRaceID() == raceId) { // If current stage race id matches the given race id
1414                 int[] riderIds = getRidersRankInStage(stage.getStageId()); // Creates new integer array
                      storing the ordered rider ids for the stage
1415                 int[] riderPoints = getRidersPointsInStage(stage.getStageId()); // Creates new integer array
                      storing the points in the stage, ordered corresponding to the riderIds array
1416                 for (int i=0; i<riderIds.length; i++){ // For i in range 0 - riderIds' length
1417                     pointsMap.put(riderIds[i], pointsMap.get(riderIds[i]) + riderPoints[i]); // Replaces the
                          current points in totalTimes for the current rider with their current points + the
                          current stage's returned points
1418                 }
1419             }
1420         }
1421         List<Map.Entry<Integer, Integer> > list = new ArrayList<Map.Entry<Integer, Integer>
              >(pointsMap.entrySet()); // Creates new List containing Map.Entry elements which contain
              pointsMap's entry set
1422
1423         // Using collections class sort method
1424         // and inside which we are using
1425         // custom comparator to compare value of map
1426         Collections.sort(list, new Comparator<Map.Entry<Integer, Integer> >() {
1427                 // Comparing two entries by value
1428                 public int compare(
1429                     Map.Entry<Integer, Integer> entry1,
1430                     Map.Entry<Integer, Integer> entry2)
1431                 {
1432                     // Substracting the entries
1433                     return entry2.getValue() - entry1.getValue(); // Return the result of the difference
                          between the values of entry1 and entry 2
1434                 }
1435             });
1436         int[] pointsList = new int[pointsMap.size()]; // Create new integer array with the size being the
              size of pointsMap
1437         int count = 0; // Set an integer count to 0
```

```
1438        for (Map.Entry<Integer, Integer> l : list) { // For each Map entry in list
1439            pointsList[count] = l.getValue(); // Stores the current entry's value in pointsList at position
                    count
1440            count++; // Increment count by 1
1441        }
1442        return pointsList; // Returns the pointsList array
1443    }
1444
1445
1446    /**
1447     * @param raceId
1448     * @return int[]
1449     * @throws IDNotRecognisedException
1450     */
1451    @Override
1452    public int[] getRidersMountainPointClassificationRank(int raceId) throws IDNotRecognisedException {
1453        assert raceId > 0 : "Race ID must be greater than 0"; // Throws an assertion exception if the raceId
                is less than 0
1454        boolean found = false; // Sets variable found to false
1455        LinkedHashMap<Integer,Integer> pointsMap = new LinkedHashMap<Integer,Integer>(); // Creates new
                LinkedHashMap called pointsMap
1456        for (Race race: Races) { // For each race in Races
1457            if (race.getId() == raceId) { // If current race's id is equal too the given race id
1458                found = true; // Sets found variable to true
1459                break;
1460            }
1461        }
1462        if (!found) { // If not found
1463            throw new IDNotRecognisedException("Race ID not recognized"); // Throws a new
                    IDNotRecognisedException with message "Race ID not recognized"
1464        }
1465        for (Rider rider : Riders){ // For each rider in Riders ArrayList
1466            pointsMap.put(rider.getId(), 0); // Gives each rider an initial value of 0 points
1467        }
1468        for (Stage stage : Stages){ // For each stage in Stages ArrayList
1469            if (stage.getRaceID() == raceId) { // If current stage race id matches the given race id
1470                int[] riderIds = getRidersRankInStage(stage.getStageId()); // Creates new integer array
                        storing the ordered rider ids for the stage
1471                int[] riderPoints = getRidersMountainPointsInStage(stage.getStageId()); // Creates new
                        integer array storing the points in the stage, ordered corresponding to the riderIds
                        array
1472                for (int i=0; i<riderIds.length; i++){ // For i in range 0 - riderIds' length
1473                    pointsMap.put(riderIds[i], pointsMap.get(riderIds[i]) + riderPoints[i]); // Replaces the
                            current points in totalTimes for the current rider with their current points + the
                            current stage's returned points
1474                }
1475            }
1476        }
1477        List<Map.Entry<Integer, Integer> > list = new ArrayList<Map.Entry<Integer, Integer>
                >(pointsMap.entrySet()); // Creates new List containing Map.Entry elements which contain
                pointsMap's entry set
1478
1479        // Using collections class sort method
1480        // and inside which we are using
1481        // custom comparator to compare value of map
```

```
        Collections.sort(list, new Comparator<Map.Entry<Integer, Integer> >() {
                // Comparing two entries by value
                public int compare(
                    Map.Entry<Integer, Integer> entry1,
                    Map.Entry<Integer, Integer> entry2)
                {
                    // Substracting the entries
                    return entry2.getValue() - entry1.getValue(); // Return the result of the difference
                        between the values of entry1 and entry 2
                }
            });
        int[] pointsList = new int[pointsMap.size()]; // Create new integer array with the size being the
             size of pointsMap
        int count = 0; // Set an integer count to 0
        for (Map.Entry<Integer, Integer> l : list) { // For each Map entry in list
            pointsList[count] = l.getValue(); // Stores the current entry's value in pointsList at position
                 count
            count++; // Increment count by 1
        }
        return pointsList; // Returns the pointsList array
    }

}
```

2 Rider.java

```
package cycling;

import java.io.Serializable;

public class Rider implements Serializable {
    private final int teamId;
    private final String name;
    private final int yearOfBirth;
    private final int id;
    private static int numberOfRiders = 0;


    public Rider(int teamId, String name, int yearOfBirth) {
        this.teamId = teamId;
        this.name = name;
        this.yearOfBirth = yearOfBirth;
        this.id = ++numberOfRiders;
    }



    /**
     * @return int
     */
    public int getTeamId() {
        return teamId;
    }

```

```
29      /**
30       * @return String
31       */
32      public String getName() {
33          return name;
34      }
35
36      /**
37       * @return int
38       */
39      public int getYearOfBirth() {
40          return yearOfBirth;
41      }
42
43      /**
44       * @return int
45       */
46      public int getId() {
47          return id;
48      }
49
50
51      public void clearNumberOfRiders() {
52          numberOfRiders = 0;
53      }
54
55      /**
56       * @param numberOfRiders
57       */
58      public void setNumberOfRiders(int numberOfRiders) {
59          Rider.numberOfRiders = numberOfRiders;
60      }
61
62
63  }
```

3 Team.java

```
1   package cycling;
2
3   import java.io.Serializable;
4
5   public class Team implements Serializable{
6       private String name;
7       private String description;
8       private int teamID;
9       private static int numberOfTeams = 0;
10
11      public Team(String name, String description) {
12          this.name = name;
13          this.description = description;
14          this.teamID = ++numberOfTeams;
15      }
16
```

```
17
18      /**
19       * @return String
20       */
21      //Getters
22      public String getName() {
23          return name;
24      }
25
26      /**
27       * @return String
28       */
29      public String getDescription() {
30          return description;
31      }
32
33      /**
34       * @return int
35       */
36      public int getTeamId() {
37          return teamID;
38      }
39
40      public void clearNumberOfTeams() {
41          numberOfTeams=0;
42      }
43
44
45      /**
46       * @param numberOfTeams
47       */
48      //setters
49      public void setNumberOfTeams( int numberOfTeams ) {
50          Team.numberOfTeams = numberOfTeams;
51      }
52
53  }
```

4 Segment.java

```
1   package cycling;
2
3   import java.io.Serializable;
4
5   public class Segment implements Comparable<Segment>, Serializable {
6       private int stageId;
7       private double location;
8       private SegmentType type;
9       private double averageGradient;
10      private double length;
11      private int id;
12      private static int numberOfSegments = 0;
13
14      public Segment(int stageId, double location, SegmentType type, double averageGradient, double length) {
```

```
15          this.stageId = stageId;
16          this.location = location;
17          this.type = type;
18          this.averageGradient = averageGradient;
19          this.length = length;
20          id = ++numberOfSegments;
21      }
22
23      public Segment(int stageId, double location, SegmentType type) {
24          this.stageId = stageId;
25          this.location = location;
26          this.type = type;
27          id = ++numberOfSegments;
28      }
29
30
31
32
33      /**
34       * @return int
35       */
36      public int getStageId() {
37          return stageId;
38      }
39
40      /**
41       * @return double
42       */
43      public double getLocation() {
44          return location;
45      }
46
47      /**
48       * @return SegmentType
49       */
50      public SegmentType getType() {
51          return type;
52      }
53
54      /**
55       * @return double
56       */
57      public double getAverageGradient() {
58          return averageGradient;
59      }
60
61      /**
62       * @return double
63       */
64      public double getLength() {
65          return length;
66      }
67
68      /**
69       * @return int
```

```
70       */
71      public int getSegmentId() {
72          return id;
73      }
74
75      /**
76       * @return int
77       */
78      public static int getNumberOfSegments() {
79          return numberOfSegments;
80      }
81
82
83
84      public void clearNumberOfSegments() {
85          numberOfSegments = 0;
86      }
87
88      /**
89       * @param numberOfSegments
90       */
91      public void setNumberOfSegments( int numberOfSegments ) {
92          Segment.numberOfSegments = numberOfSegments;
93      }
94
95
96      /**
97       * @param s
98       * @return int
99       */
100     @Override
101    public int compareTo(Segment s) {
102       return Double.compare(this.getLocation(), s.getLocation());
103    }
104 }
```

5 Stage.java

```
1  package cycling;
2
3  import java.time.LocalTime;
4  import java.io.Serializable;
5  import java.time.LocalDateTime;
6  import java.util.ArrayList;
7  import java.util.Collections;
8  import java.util.Comparator;
9  import java.util.HashMap;
10 import java.util.Iterator;
11 import java.util.LinkedHashMap;
12 import java.util.LinkedList;
13 import java.util.List;
14 import java.util.Map;
15 import java.util.Set;
16
```

```
17
18
19  public class Stage implements Comparable<Stage>, Serializable{
20      private final int raceID;
21      private final String stageName;
22      private final String description;
23      private final double length;
24      private final LocalDateTime startTime;
25      private final StageType type;
26      private boolean waitingForResults = false;
27      private final int stageId;
28      private static int numberOfStages = 0;
29
30
31      private LinkedHashMap<Integer, LocalTime[]> stageTimes = new LinkedHashMap<Integer,LocalTime[]>();
32      private LinkedHashMap<Integer, LocalTime[]> tempStageTimes = new LinkedHashMap<Integer,LocalTime[]>();
33
34
35      public Stage(int raceID, String stageName, String description, double length, LocalDateTime startTime,
            StageType type) {
36          this.raceID = raceID;
37          this.stageName = stageName;
38          this.description = description;
39          this.length = length;
40          this.startTime = startTime;
41          this.type = type;
42          stageId = ++numberOfStages;
43      }
44
45
46
47      /**
48       * @return int
49       */
50      public int getRaceID(){
51          return raceID;
52      }
53
54
55      /**
56       * @return String
57       */
58      public String getStageName(){
59          return stageName;
60      }
61
62
63      /**
64       * @return String
65       */
66      public String getDescription(){
67          return description;
68      }
69
70
```

```
71      /**
72       * @return double
73       */
74      public double getLength(){
75          return length;
76      }
77
78
79      /**
80       * @return LocalDateTime
81       */
82      public LocalDateTime getStartTime(){
83          return startTime;
84      }
85
86
87      /**
88       * @return StageType
89       */
90      public StageType getType(){
91          return type;
92      }
93
94
95      /**
96       * @return int
97       */
98      public int getStageId(){
99          return stageId;
100     }
101
102
103     /**
104      * @return boolean
105      */
106     public boolean getWaitingForResults() {
107         return waitingForResults;
108     }
109
110
111     /**
112      * @param riderId
113      * @return LocalTime[]
114      */
115     public LocalTime[] getStageTimes(int riderId){
116         return stageTimes.get(riderId);
117     }
118
119
120     /**
121      * @return LinkedHashMap<Integer, LocalTime[]>
122      */
123     public LinkedHashMap<Integer, LocalTime[]> getHashMap() {
124         return stageTimes;
125     }
```

```
126
127     /**
128      * @param riderId
129      * @return int
130      */
131     public int getRank(int riderId){
132         Set<Integer> keys = stageTimes.keySet();
133         List<Integer> listKeys = new ArrayList<Integer>( keys );
134         return listKeys.indexOf(riderId)+1;
135     }
136
137     /**
138      * @param rank
139      * @return int
140      */
141     public int getRiderIdFromRank(int rank){
142         Set<Integer> keys = stageTimes.keySet();
143         List<Integer> listKeys = new ArrayList<Integer>( keys );
144         return (int)listKeys.get(rank-1);
145     }
146
147
148
149     /**
150      * @param riderId
151      * @param timesArray
152      */
153     public void setStageTimes(int riderId, LocalTime[] timesArray) {
154         stageTimes.put(riderId, timesArray);
155     }
156
157
158     /**
159      * @param inState
160      */
161     public void setWaitingForResults(boolean inState) {
162         this.waitingForResults = inState;
163     }
164
165     public void clearNumberOfStages() {
166         numberOfStages=0;
167     }
168
169     /**
170      * @param riderId
171      */
172     public void removeRiderResults(int riderId) {
173         if (stageTimes.containsKey(riderId)) {
174             stageTimes.remove(riderId);
175         }
176     }
177
178
179     /**
180      * @param numberOfStages
```

```
181      */
182     public void setNumberOfStages( int numberOfStages ) {
183         Stage.numberOfStages = numberOfStages;
184     }
185 
186     public void sortHashMap() {
187 
188         Map<Integer, LocalTime[]> map = sortValues(stageTimes);
189         // System.out.println("After Sorting:");
190         Set set2 = map.entrySet();
191         Iterator iterator2 = set2.iterator();
192         while(iterator2.hasNext()){
193         Map.Entry me2 = (Map.Entry)iterator2.next();
194         tempStageTimes.put((Integer)me2.getKey(), (LocalTime[])me2.getValue());
195         // System.out.println("Rider ID: "+me2.getKey()+" Time:
                "+Arrays.toString((LocalTime[])me2.getValue()));
196 
197         }
198         stageTimes = tempStageTimes;
199     }
200 
201 
202     /**
203      * @param map
204      * @return HashMap
205      */
206     private static HashMap sortValues(HashMap map) {
207         List list = new LinkedList(map.entrySet());
208         //Custom Comparator
209         Collections.sort(list, new Comparator() {
210         public int compare(Object o1, Object o2) {
211             int size = ((LocalTime[])((Map.Entry) (o1)).getValue()).length;
212             // System.out.println(((Comparable) ((LocalTime[])((Map.Entry)
                    (o1)).getValue())[size-1].minusNanos(((LocalTime[])((Map.Entry)
                    (o1)).getValue())[0].toNanoOfDay())).compareTo(((LocalTime[])((Map.Entry)
                    (o2)).getValue())[size-1].minusNanos(((LocalTime[])((Map.Entry)
                    (o2)).getValue())[0].toNanoOfDay())));
213             return ((Comparable) ((LocalTime[])((Map.Entry)
                    (o1)).getValue())[size-1].minusNanos(((LocalTime[])((Map.Entry)
                    (o1)).getValue())[0].toNanoOfDay())).compareTo(((LocalTime[])((Map.Entry)
                    (o2)).getValue())[size-1].minusNanos(((LocalTime[])((Map.Entry)
                    (o2)).getValue())[0].toNanoOfDay()));
214         }
215     });
216 
217     //copying the sorted list in HashMap to preserve the iteration order
218         HashMap sortedHashMap = new LinkedHashMap();
219         for (Iterator it = list.iterator(); it.hasNext();){
220             Map.Entry entry = (Map.Entry) it.next();
221             sortedHashMap.put(entry.getKey(), entry.getValue());
222         }
223         return sortedHashMap;
224     }
225 
226 
```

```
227     /**
228      * @param s
229      * @return int
230      */
231     @Override
232    public int compareTo(Stage s) {
233       return this.getStartTime().compareTo(s.getStartTime());
234    }
235 
236     public void clearTimes() {
237         stageTimes.clear();
238         tempStageTimes.clear();
239     }
240 }
```

6 Race.java

```
1  package cycling;
2  
3  import java.io.Serializable;
4  
5  public class Race implements Serializable {
6      private final String name;
7      private final String description;
8      private final int id;
9      private static int numberOfRaces = 0;
10 
11 
12     public Race(String name, String description) {
13         this.name = name;
14         this.description = description;
15         id = ++numberOfRaces;
16 
17     }
18 
19     /**
20      * @return int
21      */
22     public int getId() {
23         return id;
24     }
25 
26 
27     /**
28      * @return String
29      */
30     public String getName() {
31         return name;
32     }
33 
34 
35     /**
36      * @return Integer
37      */
```

```
38      public Integer getNumberOfRaces() {
39          return Race.numberOfRaces;
40      }
41
42
43      /**
44       * @return String
45       */
46      public String getDescription(){
47          return description;
48      }
49
50      public void clearNumberOfRaces() {
51          numberOfRaces=0;
52      }
53
54
55      /**
56       * @param numberOfRaces
57       */
58      public void setNumberOfRaces(int numberOfRaces) {
59          Race.numberOfRaces = numberOfRaces;
60      }
61
62  }

    ;
```