1 CyclingPortal.java

```
1     public ArrayList<Race> races = new ArrayList<>();
2     public ArrayList<Team> teams = new ArrayList<>();
3     public ArrayList<Result> results = new ArrayList<>();

638   /**
639    * Checks a string, throw exception if it's null, empty or greater than 30 characters.
640    *
641    * @param name String of the name to be checked.
642    * @throws InvalidNameException If name is an empty string or if
643    *                              name length is greater than 30 chars.
644    */
645   private void checkInvalidNameException(String name) throws InvalidNameException{
646       // throws exception if name is invalid
647       if (name.trim().isEmpty() || name.length() > 30) {
648           throw new InvalidNameException(name + " is invalid!");
649       }
650   }
651
652   /**
653    * Checks if the team name is unique and doesn't already exist.
654    * Throws an exception if otherwise.
655    *
656    * @param name String of the team name to be checked.
657    * @throws IllegalNameException If the team name already exists in the system.
658    *
659    */
660   private void checkIllegalTeamName(String name) throws IllegalNameException {
661       // check for existing team name
662       for (Team i : teams) {
663           if (i.getTeamName().equals(name)) {
664               throw new IllegalNameException("Team name: '" + name + "' already exists!");
665           }
666       }
667   }
668
669   /**
670    * Checks if the race name is unique and doesn't already exist.
671    * Throws an exception if otherwise.
672    *
673    * @param raceName String of the race name to be checked.
674    * @throws IllegalNameException If the race name already exists in the system.
675    *
676    */
677   private void checkIllegalRaceName (String raceName) throws IllegalNameException{
678       // check for existing race name
679       for (Race i : races) {
680           if (i.getRaceName().equals(raceName)) {
681               throw new IllegalNameException("Race name: '" + raceName + "' already exists!");
682           }
683       }
684   }
685
```

```
686     /**
687      * Checks if length is greater than 5.0
688      *
689      * @param length Length of type double to be checked.
690      * @throws InvalidLengthException If length is less than 5.0.
691      *
692      */
693     private void checkInvalidLength (double length) throws InvalidLengthException {
694         if (length < 5.0){
695             throw new InvalidLengthException("Length must be less than 5km!");
696         }
697     }
698 
699     /**
700      * Check if location is greater than 5.0
701      *
702      * @param location Location type Double to be checked.
703      * @throws InvalidLocationException If location is less than 5.0.
704      *
705      */
706     private void checkInvalidLocation (double location) throws InvalidLocationException {
707         if (location < 5.0){
708             throw new InvalidLocationException("Location must be less than 5km!");
709         }
710     }
711 
712 
713     /**
714      * Confirms if race ID exists in the system to prevent IndexOutOfBounds exception.
715      *
716      * @param id The Race ID to be checked
717      * @throws IDNotRecognisedException If Race ID doesn't exist: is not found in the system.
718      *
719      */
720     private void checkRaceIdNotRecognised (int id) throws IDNotRecognisedException {
721         // loop through all races to check for a matching id
722         for (Race i : races){
723             if (i.getRaceId() == id){
724                 return;
725             }
726         }
727         throw new IDNotRecognisedException("Race ID: '"+ id + "' doesn't exist!");
728     }
729 
730     /**
731      * Checks if the stage name is unique/ doesn't already exist in the system.
732      *
733      * @param stageName String of the stage name to be checked
734      * @throws IllegalNameException If the stage name already exists.
735      *
736      */
737     private void checkIllegalStageName (String stageName) throws IllegalNameException{
738         // loop through races
739         for (Race i : races){
740             // loop through all stages in the current race
```

```
            for (Stage j : i.getStages()){
                // check for a match
                if (j.getStageName().equals(stageName)){
                    throw new IllegalNameException("Stage Name: '" + stageName + "' already exists!");
                }
            }
        }
    }


    /**
     * Confirms if the stage ID exists in the system to prevent IndexOutOfBounds exception.
     *
     * @param stageId The stageId to be checked.
     * @throws IDNotRecognisedException If the stageId doesn't exist.
     *
     */
    private void checkStageIdNotRecognised(int stageId) throws IDNotRecognisedException{
        // check for ID match in all races
        for (Race i : races){
            for (Stage j : i.getStages()){
                if (j.getStageId() == stageId){
                    return;
                }
            }
        }
        throw new IDNotRecognisedException("Stage ID: '" + stageId + "' doesn't exist!");
    }


    /**
     * Returns the stage state of a stageId.
     *
     * @param stageId The stageId to be checked.
     * @return String stageState of the stageId.
     *
     */
    public String checkStageState(int stageId){
        // returns stage state
        int[] ids = findAllIdsUsingStageId(stageId);

        return races.get(ids[0]).getStages().get(ids[1]).getStageState();
    }

    /**
     * Confirms that the queried stageType is NOT a time-trial.
     *
     * @param stageId The stageId to be checked.
     * @throws InvalidStageTypeException If the stageType is a time-trial.
     *
     */
    private void checkInvalidStageType(int stageId) throws InvalidStageTypeException {
        int[] ids = findAllIdsUsingStageId(stageId);

        if (races.get(ids[0]).getStages().get(ids[1]).getStageType() == StageType.TT ){
```

```
796             throw new InvalidStageTypeException("Time-trial stages cannot contain any segments!");
797         }
798     }
799
800
801     /**
802      * Confirms if the teamId exists in the system to prevent IndexOutOfBounds exception.
803      *
804      * @param teamId The teamId to be checked.
805      * @throws IDNotRecognisedException If the teamId is not found in the system.
806      *
807      */
808     public void checkTeamIdNotRecognised(int teamId) throws IDNotRecognisedException {
809         // throws exception if team id doesn't exist
810         for (Team i: teams){
811             if (i.getTeamId() == teamId){
812                 return;
813             }
814         }
815         throw new IDNotRecognisedException("Team ID: '" + teamId + "' doesn't exist!");
816     }
817
818     /**
819      * Checks if the riderName and yearOfBirth is valid before adding new rider.
820      *
821      * @param name The name of the rider to be checked.
822      * @param yearOfBirth The year of birth to be checked.
823      * @throws IllegalArgumentException If rider name is empty OR if the
824      *                                  year of birth is less than 1900
825      */
826     public void checkIllegalArgument(String name, int yearOfBirth) throws IllegalArgumentException {
827         if (name.isEmpty()) {
828             throw new IllegalArgumentException("Rider name cannot be empty!");
829         }
830         else if (yearOfBirth < 1900){
831             throw new IllegalArgumentException("Rider year of birth must be greater than 1900!");
832         }
833     }
834
835     /**
836      * Check if the riderId exists in the system.
837      *
838      * @param riderId The ID of the rider to be checked.
839      * @return Index of the team which the rider belongs to.
840      * @throws IDNotRecognisedException If rider cannot be found in the system.
841      */
842     public int checkRiderIdNotRecognised(int riderId) throws IDNotRecognisedException {
843         // throws exception if riderId doesn't exist
844         int teamPos = 0;
845         for (Team i : teams){
846             for (Rider j : i.getRiders()){
847                 if (j.getRiderId() == riderId){
848                     return teamPos;
849                 }
850             }
```

```
851             teamPos++;
852         }
853         throw new IDNotRecognisedException("Rider ID: '" + riderId + "' doesn't exist!");
854     }
855 
856     /**
857      * Check if the length of checkpoints are valid before trying to
858      * register rider results.
859      *
860      *
861      * @param stageId The stageId of which the checkpoints are t
862      * @param checkpoints The array of checkpoints to be checked.
863      * @throws InvalidCheckpointsException If the length of the checkpoints are NOT
864      *                                     +2 of the segments in that stage. The 2 other
865      *                                     times are to represent startTime and finishTime.
866      *
867      */
868     public void checkInvalidCheckpoints (int stageId, LocalTime... checkpoints) throws
            InvalidCheckpointsException{
869         // throws exception if checkpoints are not the correct length
870         for (Race i : races){
871             for (Stage j : i.getStages()){
872                 if (j.getStageId() == stageId){
873                     int criteria = j.getSegments().size() + 2;
874                     if (checkpoints.length != criteria){
875                         throw new InvalidCheckpointsException("Checkpoint length for this race must be > " +
                                criteria);
876                     }
877                 }
878             }
879         }
880     }
881 
882     /**
883      * Checks if there's already a result registered to the queried rider and stage.
884      *
885      * @param stageId The stageId to be checked
886      * @param riderId The riderId to be checked.
887      * @throws DuplicatedResultException If there are existing results for the queried
888      *                                   rider and stage.
889      *
890      */
891     public void checkDuplicatedResults (int stageId, int riderId) throws DuplicatedResultException{
892         // throws exception if there's existing results
893         for (Result i : results){
894             if (i.getResultRiderId() == riderId && i.getResultStageId() == stageId){
895                 throw new DuplicatedResultException("Results have already been registered for this rider and
                        stage!");
896             }
897         }
898     }
899 
900 
901     /**
902      * Checks if the queried segmentId exists in the system.
```

```
     *
     * @param segmentId The segmentId to be checked.
     * @return An array storing the indexes needed to access the segmentId.
     *        int[0] = race index,
     *        int[1] = stage index,
     *        int[2] = and segment index.
     * @throws IDNotRecognisedException If the segmentId doesn't exist in the system.
     *
     */
    public int[] checkSegmentIdNotRecognised(int segmentId) throws IDNotRecognisedException{
        // throws exception if existing segmentId cannot be found
        boolean found = false;

        // loop through races, stages and segments
        for (int i = 0; i < races.size(); i++){
            for (int j = 0; j < races.get(i).getStages().size(); j++){
                for (int k = 0; k < races.get(i).getStages().get(j).getSegments().size(); k++){
                    // segment found
                    if (races.get(i).getStages().get(j).getSegments().get(k).getSegmentId() == segmentId){
                        // return positions
                        return new int[]{i,j,k};
                    }
                }
            }
        }
        if (!found){
            throw new IDNotRecognisedException("Segment ID: '" + segmentId + "' doesn't exist!");
        }

        return new int[0];
    }


    /**
     * Check if there are results registered for the queried stage.
     *
     * @param stageId The stageId to be checked.
     * @return true : results exists for this stage,
     *        false : results not found for this stage.
     *
     */
    public boolean checkResultsUsingStageId(int stageId){
        // return true if there's existing results for this stage

        // loop through all results
        for (Result i : results){
            if (i.getResultStageId() == stageId){
                return true;
            }
        }
        return false;
    }


    /**
```

```
     * Check if there are results registered for the queried
     *  stage AND rider.
     *
     * @param stageId The stageId to be checked.
     * @param riderId The riderId to be checked.
     * @return true : results exist for this stage and rider,
     *         false : results doesn't exist for this stage and rider.
     *
     */
    public boolean checkResultsUsingStageIdAndRiderId(int stageId, int riderId){
        // return true if results found for this stage and rider
        for (Result i : results){
            if (i.getResultStageId() == stageId || i.getResultRiderId() == riderId){
                return true;
            }
        }
        return false;
    }

    /**
     * Get the indexes of all results registered to the queried stage.
     *
     * @param stageId The stageId to be queried.
     * @return Array containing the indexes of results registered to
     *         the queried stage.
     * @throws IDNotRecognisedException If there are no results registered
     *                                  for this stage.
     *
     */
    public int[] findResultsIndexArrayUsingStageId(int stageId) throws IDNotRecognisedException{
        // initialize empty arrayList
        ArrayList<Integer> resultsPos = new ArrayList<>();
        boolean found = false;
        int count = 0;

        // find matching ids in results
        for (Result i : results){
            if (i.getResultStageId() == stageId){
                // add index to arrayList
                resultsPos.add(count);
                found = true;
            }
            count++;
        }

        if (!found){
            throw new IDNotRecognisedException("Results not found for stage ID: " + stageId);
        }

        // indexes arrayList to simple int
        int[] results = new int[resultsPos.size()];
        for (int i = 0; i < resultsPos.size(); i++){
            results[i] = resultsPos.get(i);
        }
```

```
1013            return results;
1014        }
1015
1016        /**
1017         * Get the index of the result registered to the queried
1018         * stage AND rider.
1019         *
1020         * @param stageId The stageId to be queried.
1021         * @param riderId The riderId to be queried.
1022         * @return The index of the results registered to the stage
1023         *         and rider.
1024         * @throws IDNotRecognisedException If no results are found for the
1025         *                                  queried stage and rider.
1026         *
1027         */
1028        public int findResultsIndexUsingStageIdAndRiderId (int stageId, int riderId) throws
                IDNotRecognisedException{
1029            // return results index for this stage and rider
1030
1031            int resultPos = 0;
1032            boolean found = false;
1033
1034            // search for a match in results
1035            for (Result i : results){
1036                if (i.getResultStageId() == stageId && i.getResultRiderId() == riderId){
1037                    return resultPos;
1038                }
1039                resultPos++;
1040            }
1041
1042            if (!found){
1043                throw new IDNotRecognisedException("Results not found for these IDs!");
1044            }
1045
1046            return 0;
1047        }
1048
1049        //========= Non-exception related functions ================================================================
1050
1051        /**
1052         * Get the index of this raceId.
1053         *
1054         * @param raceId The raceId to be queried.
1055         * @return The index of the raceId in the races ArrayList.
1056         *
1057         */
1058        public int findRaceId (int raceId){
1059            // returns index for this raceId
1060
1061            int racePos = 0;
1062            for (Race i : races){
1063                if(i.getRaceId() == raceId){
1064                    return racePos;
1065                }
1066                racePos++;
```

```
        }
        return 0;
    }


    /**
     * Get the index of the queried riderId.
     *
     * @param riderId The riderId to be queried.
     * @return The index of the riderId in the riders ArrayList.
     *
     */
    public int findRiderId(int riderId){
        // returns rider index in riders ArrayList
        for (Team i : teams){
            int riderPos = 0;
            for (Rider j : i.getRiders()){
                if (j.getRiderId() == riderId){
                    return riderPos;
                }
                riderPos++;
            }
        }
        return 0;
    }


    /**
     * Get the indexes necessary to access the queried stageId.
     *
     * @param stageId The stageId to be queried.
     * @return An array of the indexes to access the queried stageId.
     *         int[0] = race index,
     *         int[1] = stage index.
     *
     */
    private int[] findAllIdsUsingStageId(int stageId){
        // returns the race and stage index of this stage

        int racePos = 0;
        for (Race i : races){
            int stagePos = 0;
            for (Stage j : i.getStages()){
                if (j.getStageId() == stageId){
                    return new int[]{racePos,stagePos};
                }
                stagePos++;
            }
            racePos++;
        }
        return new int[]{0,0};
    }

    /**
     * Calculates the total length of the race :
```

```
     * The sum of all stage lengths in that race.
     *
     * Called whenever a new stage is added to the race.
     *
     * @param raceId The raceId of the calculation.
     * @return The total length of the race.
     *
     */
    public double calculateTotalRaceLength(int raceId){
        int racePos = findRaceId(raceId);
        double total = 0;

        // sum of all stage lengths
        for (Stage j : races.get(racePos).getStages()){
            total += j.getStageLength();
        }

        return total;
    }

    /**
     * Get the indexes of results registered to the queried race.
     *
     * @param raceId The raceId to be queried.
     * @return An array of indexes of results registered to this race.
     *
     */
    public int[] findResultsIndexArrayUsingRaceId (int raceId){
        // returns array of results indexes for this race
        int racePos = findRaceId(raceId);

        // use raceId to get all stageIds
        ArrayList<Stage> stages = races.get(racePos).getStages();
        int[] stageIds = new int[stages.size()];
        //ArrayList<Integer> resultPos = new ArrayList<>();

        int counter = 0;
        for (Stage i : stages){
            stageIds[counter] = i.getStageId();
            counter++;
        }

        // use stageId array to find all matching results
        counter = 0;
        ArrayList<Integer> tempArray = new ArrayList<>();

        for (Result i : results){
            // check for matches in the array of stageIds
            for (int j = 0; j < stageIds.length; j++){
                if (i.getResultStageId() == stageIds[j]){
                    tempArray.add(counter);
                }
            }
            counter++;
        }
```

```
1177
1178        // to simple array
1179        int[] idArray = new int[tempArray.size()];
1180        for (int i = 0; i < tempArray.size(); i++){
1181            idArray[i] = tempArray.get(i);
1182        }
1183
1184        return idArray;
1185    }
1186
1187 }
```

2 Race.java

```
1  package cycling;
2
3  import java.io.Serializable;
4  import java.util.ArrayList;
5
6  /**
7   * Race class:
8   * Used to hold data of Races
9   * e.g. name, description, length
10  *
11  * Includes get/set methods
12  *
13  * @author Daphne Yap
14  * @version 1.0
15  *
16  */
17
18 public class Race implements Serializable {
19     private final String raceName;
20     private final String raceDesc;
21     private double length;
22     private final ArrayList<Stage> stages = new ArrayList<>();
23     private final int raceId;
24     private static int raceCount;
25
26     // constructor to insert all details in the object
27     Race(String raceName, String raceDesc) {
28         this.raceName = raceName;
29         this.raceDesc = raceDesc;
30         this.raceId = ++raceCount;
31     }
32
33     public String getRaceName() {
34         return raceName;
35     }
36
37     public String getRaceDesc(){
38         return raceDesc;
39     }
40
```

```
    public double getRaceLength() {
        return length;
    }

    public ArrayList<Stage> getStages() {
        return this.stages;
    }

    public int getRaceId(){
        return this.raceId;
    }

    public void addStage(Stage stage){
        stages.add(stage);
    }

    public void setRaceLength(double length){
        this.length = length;
    }

}
```

3 Stage.java

```
package cycling;

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

/**
 * Stage class:
 * Used to hold data of Stages
 * e.g. name, description, length
 *
 * Includes get/set methods
 *
 * @author Daphne Yap
 * @version 1.0
 *
 */

public class Stage implements Serializable {
    private int raceId;
    private String stageName;
    private String stageDesc;
    private double length;
    private LocalDateTime startTime;
    private StageType type;
    private int stageId;
    private static int stageCount;
    private String stageState;
    private ArrayList<Segment> segments = new ArrayList<>();

```

```
31      // constructor to insert all details in the object
32      Stage (int raceId, String stageName, String stageDesc, double length, LocalDateTime startTime,
            StageType type){
33          this.raceId = raceId;
34          this.stageName = stageName;
35          this.stageDesc = stageDesc;
36          this.length = length;
37          this.startTime = startTime;
38          this.type = type;
39          this.stageId = ++stageCount;
40          this.stageState = "in preparation";
41      }
42
43      public int getRaceId(){
44          return this.raceId;
45      }
46
47      public String getStageName(){
48          return this.stageName;
49      }
50
51      public String getStageDesc(){
52          return this.stageDesc;
53      }
54
55      public double getStageLength(){
56          return this.length;
57      }
58
59      public LocalDateTime getStageStartTime(){
60          return this.startTime;
61      }
62
63      public StageType getStageType(){
64          return this.type;
65      }
66
67      public int getStageId(){
68          return this.stageId;
69      }
70
71      public String getStageState(){
72          return this.stageState;
73      }
74
75      public void addSegment (Segment segment){
76          this.segments.add(segment);
77      }
78
79      public ArrayList<Segment> getSegments() {
80          return this.segments;
81      }
82
83      public void setStageState(String state){
84          this.stageState = state;
```

```
    }
}
```

4 Segment.java

```
package cycling;

import java.io.Serializable;

/**
 * Segment class:
 * Used to hold all data related to Segments
 * e.g. stageId, location, SegmentType
 *
 * Includes get/set methods
 *
 * @author Daphne Yap
 * @version 1.0
 *
 */

public class Segment implements Serializable {
    private final int stageId;
    private final double location;
    private final SegmentType type;
    private double averageGradient;
    private double length;
    private int segmentId;
    private static int segmentCount;

    // 1st constructor: for sprints
    Segment(int stageId, double location, SegmentType type){
        this.stageId = stageId;
        this.location = location;
        this.type = type;
        this.segmentId = ++segmentCount;
    }

    // 2nd constructor: for climbs, takes 2 more arguments
    // averageGradient and length
    Segment(int stageId, double location, SegmentType type, double averageGradient, double length){
        this.stageId = stageId;
        this.location = location;
        this.type = type;
        this.averageGradient = averageGradient;
        this.length = length;
        this.segmentId = ++segmentCount;
    }

    public int getSegmentStageId(){
        return this.stageId;
    }

    public double getSegmentLocation(){
```

```
50          return this.location;
51      }
52
53      public SegmentType getSegmentType(){
54          return this.type;
55      }
56
57      public double getAverageGradient(){
58          return this.averageGradient;
59      }
60
61      public double getSegmentLength(){
62          return this.length;
63      }
64
65      public int getSegmentId(){
66          return this.segmentId;
67      }
68
69  }
```

5 Team.java

```
1   package cycling;
2
3   import java.awt.image.AreaAveragingScaleFilter;
4   import java.io.Serializable;
5   import java.util.ArrayList;
6
7   /**
8    * Team class:
9    * Used to hold all data related to Team
10   * e.g. team name, description, teamId
11   *
12   * Includes get/set methods
13   *
14   * @author Daphne Yap
15   * @version 1.0
16   *
17   */
18
19  public class Team implements Serializable {
20      private String teamName;
21      private String teamDesc;
22      private int teamId;
23      private static int teamCount;
24      private ArrayList<Rider> riders = new ArrayList<>();
25
26      // constructor to insert all details in the object
27      Team(String teamName, String teamDesc){
28          this.teamName = teamName;
29          this.teamDesc = teamDesc;
30          this.teamId = ++teamCount;
31      }
```

```
32
33      public String getTeamName(){
34          return teamName;
35      }
36
37      public String getTeamDesc(){
38          return teamDesc;
39      }
40
41      public int getTeamId(){
42          return teamId;
43      }
44
45      public ArrayList<Rider> getRiders() {
46          return this.riders;
47      }
48
49  }
```

6 Rider.java

```
1   package cycling;
2
3   import java.io.Serializable;
4
5   /**
6    * Rider class:
7    * Used to hold data related to Riders
8    * e.g. name, id, year of birth.
9    *
10   * Includes get/set methods
11   *
12   * @author Daphne Yap
13   * @version 1.0
14   *
15   */
16
17  public class Rider implements Serializable {
18      private String riderName;
19      private int riderYob;
20      private int riderId;
21      private static int riderCount;
22
23      // constructor to insert all details in the object
24      Rider(String riderName, int riderYob){
25          this.riderName = riderName;
26          this.riderYob = riderYob;
27          this.riderId = ++riderCount;
28      }
29
30      public String getRiderName(){
31          return this.riderName;
32      }
33
```

```
34      public int getRiderYob(){
35          return this.riderYob;
36      }
37
38      public int getRiderId() {
39          return this.riderId;
40      }
41  }
```

7 Result.java

```
1   package cycling;
2
3   import java.io.Serializable;
4   import java.util.*;
5   import java.time.LocalTime;
6
7   /**
8    * Result class:
9    * Used to hold all data related to Results
10   * e.g. stageId, description, elapsed time
11   *
12   *
13   * Includes get/set methods
14   *
15   * @author Daphne Yap
16   * @version 1.0
17   *
18   */
19
20  public class Result implements Serializable {
21      private int stageId;
22      private int riderId;
23      private LocalTime elapsedTime;
24      private LocalTime[] checkpoints;
25      private ArrayList<Integer> points;
26
27      // constructor to insert all details in the object
28      Result(int stageId, int riderId, LocalTime elapsedTime, LocalTime... checkpoints) {
29          this.stageId = stageId;
30          this.riderId = riderId;
31          this.elapsedTime = elapsedTime;
32          this.checkpoints = checkpoints;
33      }
34
35      public int getResultStageId() {
36          return this.stageId;
37      }
38
39      public int getResultRiderId() {
40          return this.riderId;
41      }
42
43      public LocalTime getElapsedTime() {
```

```
44          return this.elapsedTime;
45      }
46
47      public LocalTime[] getCheckpoints() {
48          return this.checkpoints;
49      }
50
51      public ArrayList<Integer> getResultPoints() {
52          return this.points;
53      }
54  }
```