```java

    /**
     * Checks a string, throw exception if it's null, empty or greater than 30 characters.
     *
     * @param name String of the name to be checked.
     * @throws InvalidNameException If name is an empty string or if
     *                              name length is greater than 30 chars.
     */
    private void checkInvalidNameException(String name) throws InvalidNameException{
        // throws exception if name is invalid
        if (name.trim().isEmpty() || name.length() > 30) {
            throw new InvalidNameException(name + " is invalid!");
        }
    }

    /**
     * Checks if the team name is unique and doesn't already exist.
     * Throws an exception if otherwise.
     *
     * @param name String of the team name to be checked.
     * @throws IllegalNameException If the team name already exists in the system.
     *
     */
    private void checkIllegalTeamName(String name) throws IllegalNameException {
        // check for existing team name
        for (Team i : teams) {
            if (i.getTeamName().equals(name)) {
                throw new IllegalNameException("Team name: '" + name + "' already exists!");
            }
        }
    }

    /**
     * Checks if the race name is unique and doesn't already exist.
     * Throws an exception if otherwise.
     *
     * @param raceName String of the race name to be checked.
     * @throws IllegalNameException If the race name already exists in the system.
     *
     */
    private void checkIllegalRaceName (String raceName) throws IllegalNameException{
        // check for existing race name
        for (Race i : races) {
            if (i.getRaceName().equals(raceName)) {
                throw new IllegalNameException("Race name: '" + raceName + "' already exists
            }
        }
    }

    /**
     * Checks if length is greater than 5.0
     *
     * @param length Length of type double to be checked.
     * @throws InvalidLengthException If length is less than 5.0.
     *
     */
    private void checkInvalidLength (double length) throws InvalidLengthException {
        if (length < 5.0){
            throw new InvalidLengthException("Length must be less than 5km!");
        }
    }

    /**
     * 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
741             for (Stage j : i.getStages()){
742                 // check for a match
743                 if (j.getStageName().equals(stageName)){
744                     throw new IllegalNameException("Stage Name: '" + stageName + "' already e
745                 }
746             }
747         }
748     }
749 
750 
751     /**
752      * Confirms if the stage ID exists in the system to prevent IndexOutOfBounds exception.
753      *
754      * @param stageId The stageId to be checked.
755      * @throws IDNotRecognisedException If the stageId doesn't exist.
756      *
757      */
758     private void checkStageIdNotRecognised(int stageId) throws IDNotRecognisedException{
759         // check for ID match in all races
760         for (Race i : races){
761             for (Stage j : i.getStages()){
762                 if (j.getStageId() == stageId){
763                     return;
764                 }
```

```
765             }
766         }
767         throw new IDNotRecognisedException("Stage ID: '" + stageId + "' doesn't exist!");
768     }
769 
770 
771     /**
772      * Returns the stage state of a stageId.
773      *
774      * @param stageId The stageId to be checked.
775      * @return String stageState of the stageId.
776      *
777      */
778     public String checkStageState(int stageId){
779         // returns stage state
780         int[] ids = findAllIdsUsingStageId(stageId);
781 
782         return races.get(ids[0]).getStages().get(ids[1]).getStageState();
783     }
784 
785     /**
786      * Confirms that the queried stageType is NOT a time-trial.
787      *
788      * @param stageId The stageId to be checked.
789      * @throws InvalidStageTypeException If the stageType is a time-trial.
790      *
791      */
792     private void checkInvalidStageType(int stageId) throws InvalidStageTypeException {
793         int[] ids = findAllIdsUsingStageId(stageId);
794 
795         if (races.get(ids[0]).getStages().get(ids[1]).getStageType() == StageType.TT ){
796             throw new InvalidStageTypeException("Time-trial stages cannot contain any segment
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 IllegalArgumentExce
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 Invali
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 ra
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 DuplicatedResultExce
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
896             }
897         }
898     }
899 
900 
901     /**
902      * Checks if the queried segmentId exists in the system.
903      *
904      * @param segmentId The segmentId to be checked.
905      * @return An array storing the indexes needed to access the segmentId.
906      *         int[0] = race index,
907      *         int[1] = stage index,
908      *         int[2] = and segment index.
909      * @throws IDNotRecognisedException If the segmentId doesn't exist in the system.
910      *
911      */
912     public int[] checkSegmentIdNotRecognised(int segmentId) throws IDNotRecognisedException{
913         // throws exception if existing segmentId cannot be found
914         boolean found = false;
915 
916         // loop through races, stages and segments
917         for (int i = 0; i < races.size(); i++){
918             for (int j = 0; j < races.get(i).getStages().size(); j++){
919                 for (int k = 0; k < races.get(i).getStages().get(j).getSegments().size(); k++
920                     // segment found
921                     if (races.get(i).getStages().get(j).getSegments().get(k).getSegmentId() =
922                         // return positions
923                         return new int[]{i,j,k};
924                     }
925                 }
926             }
927         }
928         if (!found){
929             throw new IDNotRecognisedException("Segment ID: '" + segmentId + "' doesn't exist
930         }
931 
932         return new int[0];
933     }
934 
935 
936     /**
937      * Check if there are results registered for the queried stage.
938      *
939      * @param stageId The stageId to be checked.
940      * @return true : results exists for this stage,
941      *         false : results not found for this stage.
942      *
943      */
944     public boolean checkResultsUsingStageId(int stageId){
945         // return true if there's existing results for this stage
946 
947         // loop through all results
948         for (Result i : results){
949             if (i.getResultStageId() == stageId){
950                 return true;
951             }
952         }
953         return false;
954     }
955 
956 
```

```
957     /**
958      * Check if there are results registered for the queried
959      *  stage AND rider.
960      *
961      * @param stageId The stageId to be checked.
962      * @param riderId The riderId to be checked.
963      * @return true : results exist for this stage and rider,
964      *         false : results doesn't exist for this stage and rider.
965      *
966      */
967     public boolean checkResultsUsingStageIdAndRiderId(int stageId, int riderId){
968         // return true if results found for this stage and rider
969         for (Result i : results){
970             if (i.getResultStageId() == stageId || i.getResultRiderId() == riderId){
971                 return true;
972             }
973         }
974         return false;
975     }
976 
977     /**
978      * Get the indexes of all results registered to the queried stage.
979      *
980      * @param stageId The stageId to be queried.
981      * @return Array containing the indexes of results registered to
982      *         the queried stage.
983      * @throws IDNotRecognisedException If there are no results registered
984      *                                  for this stage.
985      *
986      */
987     public int[] findResultsIndexArrayUsingStageId(int stageId) throws IDNotRecognisedExcept
988         // initialize empty arrayList
989         ArrayList<Integer> resultsPos = new ArrayList<>();
990         boolean found = false;
991         int count = 0;
992 
993         // find matching ids in results
994         for (Result i : results){
995             if (i.getResultStageId() == stageId){
996                 // add index to arrayList
997                 resultsPos.add(count);
998                 found = true;
999             }
1000            count++;
1001        }
1002
1003        if (!found){
1004            throw new IDNotRecognisedException("Results not found for stage ID: " + stageId)
1005        }
1006
1007        // indexes arrayList to simple int
1008        int[] results = new int[resultsPos.size()];
1009        for (int i = 0; i < resultsPos.size(); i++){
1010            results[i] = resultsPos.get(i);
1011        }
1012
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 IDNo
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++;
1067         }
1068         return 0;
1069     }
1070
1071
1072     /**
1073      * Get the index of the queried riderId.
1074      *
1075      * @param riderId The riderId to be queried.
1076      * @return The index of the riderId in the riders ArrayList.
1077      *
1078      */
1079     public int findRiderId(int riderId){
1080         // returns rider index in riders ArrayList
1081         for (Team i : teams){
1082             int riderPos = 0;
1083             for (Rider j : i.getRiders()){
1084                 if (j.getRiderId() == riderId){
```

```
1085                    return riderPos;
1086                }
1087                riderPos++;
1088            }
1089        }
1090        return 0;
1091    }
1092
1093
1094    /**
1095     * Get the indexes necessary to access the queried stageId.
1096     *
1097     * @param stageId The stageId to be queried.
1098     * @return An array of the indexes to access the queried stageId.
1099     *         int[0] = race index,
1100     *         int[1] = stage index.
1101     *
1102     */
1103    private int[] findAllIdsUsingStageId(int stageId){
1104        // returns the race and stage index of this stage
1105
1106        int racePos = 0;
1107        for (Race i : races){
1108            int stagePos = 0;
1109            for (Stage j : i.getStages()){
1110                if (j.getStageId() == stageId){
1111                    return new int[]{racePos,stagePos};
1112                }
1113                stagePos++;
1114            }
1115            racePos++;
1116        }
1117        return new int[]{0,0};
1118    }
1119
1120    /**
1121     * Calculates the total length of the race :
1122     * The sum of all stage lengths in that race.
1123     *
1124     * Called whenever a new stage is added to the race.
1125     *
1126     * @param raceId The raceId of the calculation.
1127     * @return The total length of the race.
1128     *
1129     */
1130    public double calculateTotalRaceLength(int raceId){
1131        int racePos = findRaceId(raceId);
1132        double total = 0;
1133
1134        // sum of all stage lengths
1135        for (Stage j : races.get(racePos).getStages()){
1136            total += j.getStageLength();
1137        }
1138
1139        return total;
1140    }
1141
1142    /**
1143     * Get the indexes of results registered to the queried race.
1144     *
1145     * @param raceId The raceId to be queried.
1146     * @return An array of indexes of results registered to this race.
1147     *
1148     */
```

```
1149     public int[] findResultsIndexArrayUsingRaceId (int raceId){
1150         // returns array of results indexes for this race
1151         int racePos = findRaceId(raceId);
1152 
1153         // use raceId to get all stageIds
1154         ArrayList<Stage> stages = races.get(racePos).getStages();
1155         int[] stageIds = new int[stages.size()];
1156         //ArrayList<Integer> resultPos = new ArrayList<>();
1157 
1158         int counter = 0;
1159         for (Stage i : stages){
1160             stageIds[counter] = i.getStageId();
1161             counter++;
1162         }
1163 
1164         // use stageId array to find all matching results
1165         counter = 0;
1166         ArrayList<Integer> tempArray = new ArrayList<>();
1167 
1168         for (Result i : results){
1169             // check for matches in the array of stageIds
1170             for (int j = 0; j < stageIds.length; j++){
1171                 if (i.getResultStageId() == stageIds[j]){
1172                     tempArray.add(counter);
1173                 }
1174             }
1175             counter++;
1176         }
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 }
1188 
```