Rally Championships

Report

Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software

Grupo T11_1:
Diogo Serra Duque - 201406274 - 201406274@fe.up.pt
Renato Abreu - 201403377 - up201403377@fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

January 3, 2018

Contents

1 Informal system description and list of requirements

1.1 Informal system description

The main focus of our system is to represent and model all the information necessary to manage and visualize Rally Championships, and all the details related to it.

A Rally Championship is a rallying series, consisting of several rallies, culminating with a champion driver and a team.

Our system allows the management of teams, sponsors, manufacturers, cars and respective drivers that are participating on a Rally Championship.

A rally, held on a defined location, features a number (typically 15 to 25) of timed sections - known as special-stages -, which are based on straightforward speed over closed roads. Normally, it's organized throughout several days, on the most different surfaces like asphalt mountain, rough forest tracks, ice and snow or desert sand. Because of this, competitors visit a service park at pre-determined points to subject the cars to mechanical work.

The goal of each driver is to complete each stage as quickly as possible. On the other hand, the driver that completes all the stages in the shortest time is the winner of that rally.

As a result, our system must be capable of manage the information of each rally and respective itinerary (service parks, special stage, distances, terrain, etc). Whenever the rally is happening, it's also necessary to register the statistics (time, average speed, penalties) of each driver in a given stage, which consequently defines the leader board. Automatically, after each Rally Championship, it defines the respective global leader board.

1.2 List of requirements

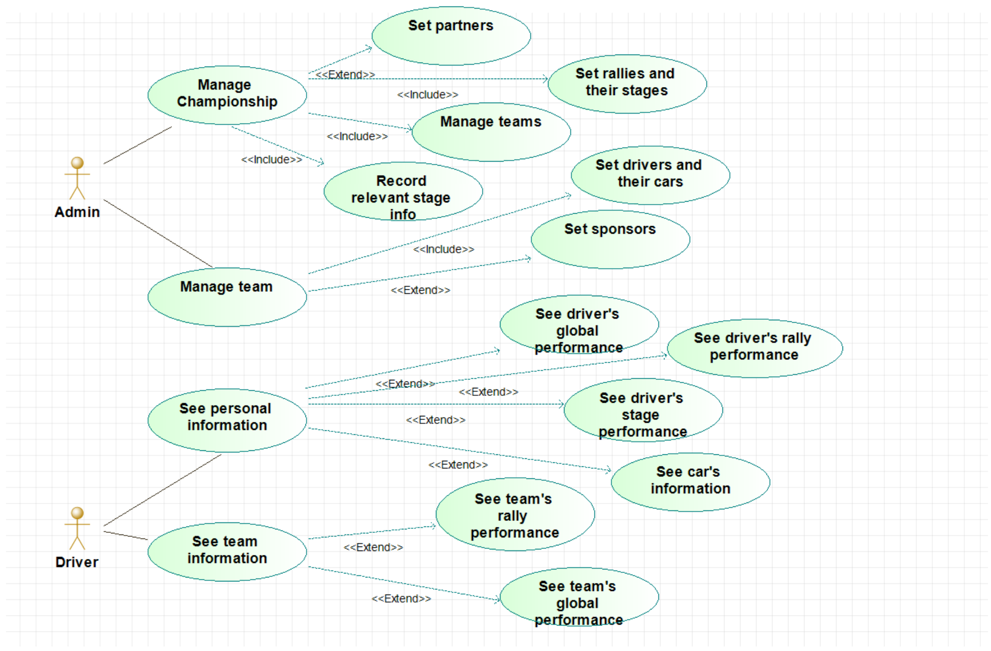
Id	Priority	Description
R1	Mandatory	As an Admin, I want to manage championships, the participating teams and all related personnel.
R2	Mandatory	As an Admin, I want to manage rallies and the respective itinerary.
R3	Mandatory	As an Admin, I want to manage the statistics of a rally, by updating the performance of each team and drivers.
R4	Mandatory	As an User, I want to access all the available championships.
R5	Mandatory	As an User, I want to check which teams and drivers participated in a given championship as well as their performances.
R6	Mandatory	As an User, I want to analyze the details of a rally, its special stages and itinerary.
R7	Mandatory	As an User, I want to access and analyze the leader board of a rally.

Table 1: Requirements

These requirements are directly translated onto use cases as shown next.

2 Visual UML model

2.1 Use case model



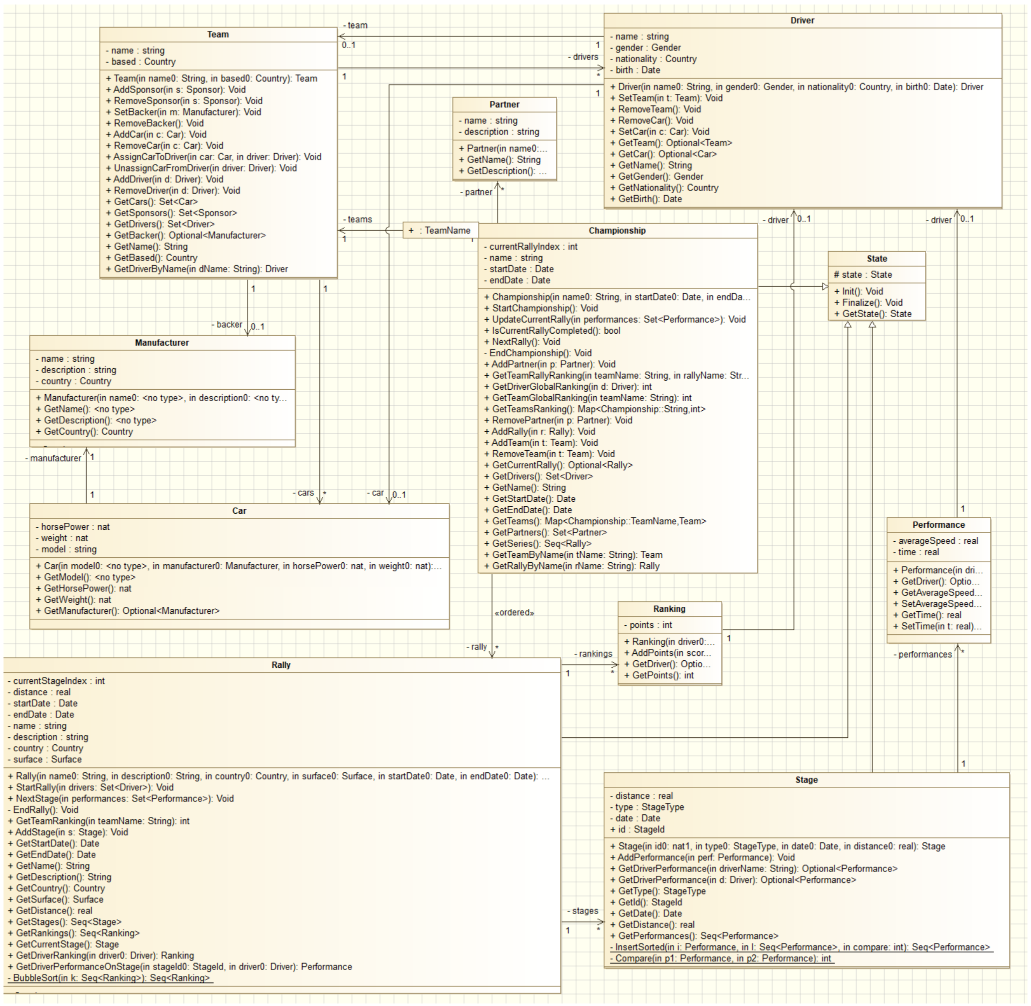
Scenario	Setup a championship
Description	Scenario containing everything that happen before a championship starts.
Steps	• Add/remove partners • Add rallies • Add teams
Pre-conditions	None
Post-conditions	• There must be teams and rallies in the championship. • Teams cannot have the same name. • Rallies cannot have the same name.

Scenario	Setup a team
Description	Scenario containing everything that happens to a team before a championship starts.
Steps	1. Populate team • Add drivers • Add cars • Add sponsors 2. Assign drivers to cars
Pre-conditions	None
Post-conditions	• Sponsors cannot have the same name. • Championship must have teams and rallies.

Scenario	**Set up rally**
Description	Scenario containing everything that happens to a rally before a championship starts.
Steps	• Add stages
Pre-conditions	None
Post-conditions	• Rally must contain stage(s).

Scenario	**Record stage info**
Description	Scenario containing what happens at the end of a given stage from a certain rally.
Steps	• Add driver's performances.
Pre-conditions	• Rally's state must be *Occurring*. • Stage's state must be *Occurring*.
Post-conditions	• All driver's performances must have been added to the stage. • Stage's state must be *Completed*.

2.2 Class model



Team
- name : string
- based : Country
+ Team(in name0: String, in based0: Country): Team
+ AddSponsor(in s: Sponsor): Void
+ RemoveSponsor(in s: Sponsor): Void
+ SetBacker(in m: Manufacturer): Void
+ RemoveBacker(): Void
+ AddCar(in c: Car): Void
+ RemoveCar(in c: Car): Void
+ AssignCarToDriver(in car: Car, in driver: Driver): Void
+ UnassignCarFromDriver(in driver: Driver): Void
+ AddDriver(in d: Driver): Void
+ RemoveDriver(in d: Driver): Void
+ GetCars(): Set<Car>
+ GetSponsors(): Set<Sponsor>
+ GetDrivers(): Set<Driver>
+ GetBacker(): Optional<Manufacturer>
+ GetName(): String
+ GetBased(): Country
+ GetDriverByName(in dName: String): Driver
Driver
- name : string
- gender : Gender
- nationality : Country
- birth : Date
+ Driver(in name0: String, in gender0: Gender, in nationality0: Country, in birth0: Date): Driver
+ SetTeam(in t: Team): Void
+ RemoveTeam(): Void
+ RemoveCar(): Void
+ SetCar(in c: Car): Void
+ GetTeam(): Optional<Team>
+ GetCar(): Optional<Car>
+ GetName(): String
+ GetGender(): Gender
+ GetNationality(): Country
+ GetBirth(): Date
Partner
- name : string
- description : string
+ Partner(in name0:...
+ GetName(): String
+ GetDescription(): ...
Championship
- currentRallyIndex : int
- name : string
- startDate : Date
- endDate : Date
+ Championship(in name0: String, in startDate0: Date, in endDa...
+ StartChampionship(): Void
+ UpdateCurrentRally(in performances: Set<Performance>): Void
+ IsCurrentRallyCompleted(): bool
+ NextRally(): Void
- EndChampionship(): Void
+ AddPartner(in p: Partner): Void
+ GetTeamRallyRanking(in teamName: String, in rallyName: Str...
+ GetDriverGlobalRanking(in d: Driver): int
+ GetTeamGlobalRanking(in teamName: String): int
+ GetTeamsRanking(): Map<Championship::String,int>
+ RemovePartner(in p: Partner): Void
+ AddRally(in r: Rally): Void
+ AddTeam(in t: Team): Void
+ RemoveTeam(in t: Team): Void
+ GetCurrentRally(): Optional<Rally>
+ GetDrivers(): Set<Driver>
+ GetName(): String
+ GetStartDate(): Date
+ GetEndDate(): Date
+ GetTeams(): Map<Championship::TeamName,Team>
+ GetPartners(): Set<Partner>
+ GetSeries(): Seq<Rally>
+ GetTeamByName(in tName: String): Team
+ GetRallyByName(in rName: String): Rally
+ : TeamName
State
# state : State
+ Init(): Void
+ Finalize(): Void
+ GetState(): State
Manufacturer
- name : string
- description : string
- country : Country
+ Manufacturer(in name0: <no type>, in description0: <no ty...
+ GetName(): <no type>
+ GetDescription(): <no type>
+ GetCountry(): Country
Car
- horsePower : nat
- weight : nat
- model : string
+ Car(in model0: <no type>, in manufacturer0: Manufacturer, in horsePower0: nat, in weight0: nat):...
+ GetModel(): <no type>
+ GetHorsePower(): nat
+ GetWeight(): nat
+ GetManufacturer(): Optional<Manufacturer>
Ranking
- points : int
+ Ranking(in driver0:...
+ AddPoints(in scor...
+ GetDriver(): Optio...
+ GetPoints(): int
Performance
- averageSpeed : real
- time : real
+ Performance(in dri...
+ GetDriver(): Optio...
+ GetAverageSpeed...
+ SetAverageSpeed...
+ GetTime(): real
+ SetTime(in t: real)...
Rally
- currentStageIndex : int
- distance : real
- startDate : Date
- endDate : Date
- name : string
- description : string
- country : Country
- surface : Surface
+ Rally(in name0: String, in description0: String, in country0: Country, in surface0: Surface, in startDate0: Date, in endDate0: Date): ...
+ StartRally(in drivers: Set<Driver>): Void
+ NextStage(in performances: Set<Performance>): Void
- EndRally(): Void
+ GetTeamRanking(in teamName: String): int
+ AddStage(in s: Stage): Void
+ GetStartDate(): Date
+ GetEndDate(): Date
+ GetName(): String
+ GetDescription(): String
+ GetCountry(): Country
+ GetSurface(): Surface
+ GetDistance(): real
+ GetStages(): Seq<Stage>
+ GetRankings(): Seq<Ranking>
+ GetCurrentStage(): Stage
+ GetDriverRanking(in driver0: Driver): Ranking
+ GetDriverPerformanceOnStage(in stageId0: StageId, in driver0: Driver): Performance
- BubbleSort(in k: Seq<Ranking>): Seq<Ranking>
Stage
- distance : real
- type : StageType
- date : Date
+ id : StageId
+ Stage(in id0: nat1, in type0: StageType, in date0: Date, in distance0: real): Stage
+ AddPerformance(in perf: Performance): Void
+ GetDriverPerformance(in driverName: String): Optional<Performance>
+ GetDriverPerformance(in d: Driver): Optional<Performance>
+ GetType(): StageType
+ GetId(): StageId
+ GetDate(): Date
+ GetDistance(): real
+ GetPerformances(): Seq<Performance>
- InsertSorted(in i: Performance, in l: Seq<Performance>, in compare: int): Seq<Performance>
- Compare(in p1: Performance, in p2: Performance): int
- team
0..1
- drivers
1
*
- partner
*
- teams
1
- driver
0..1
- backer
0..1
- manufacturer
- cars
- car
«ordered»
- rally
- rankings
- performances
- stages

Class	Description
Championship	Defines a championship, composed by several rallies.
Partner	Defines a championship partner.
Team	Defines a team participating in a championship.
Sponsor	Defines a team's sponsor.
Driver	Defines a driver, which will eventually belong to a team.
Car	Defines a car, which will eventually belong to a team.
Manufacturer	Defines a car manufacturer.
Rally	Defines a rally, composed of 1 or more stages.
Ranking	Defines a driver's performance on a rally.
Stage	Defines a stage where drivers will run.
Performance	Defines a driver's performance on a stage.
State	Superclass of Championship, Rally and Stage. Defines the state in which one of these events is in.
DateUtils	Defines several utilities for dates.

Table 2: Classes

Test Class	**Description**
MyTestCase	Superclass for test classes; defines assertEquals, assertTrue and assertFalse.
MyTestRunner	Calls all the test/usage scenarios and test cases.
ChampionshipTest	Defines tests to the Championship class.
TeamTest	Defines tests to the Team class.
SponsorTest	Defines tests to the Sponsor class.
DriverTest	Defines tests to the Driver class.
CarTest	Defines tests to the Car class.
ManufacturerTest	Defines tests to the Manufacturer class.
RallyTest	Defines tests to the Rally class.
StageTest	Defines tests to the Stage class.
PerformanceTest	Defines tests to the Performance class.
UseCasesTest	Defines and tests a possible scenario for the full usage of this project.

Table 3: Test Classes

3 Formal VDM++ model

3.1 Car

```
class Car

types
 public String = seq of char;

instance variables
  private manufacturer: [Manufacturer];
  private model: String;
  private horsePower: nat;
  private weight: nat;

operations
 /**
  * Instantiates a car instance.
  */

 public Car: String * Manufacturer * nat * nat ==> Car
  Car(model0, manufacturer0, horsePower0, weight0) == (
    model := model0;
    manufacturer := manufacturer0;
    horsePower := horsePower0;
    weight := weight0;
    return self;
  )
  pre horsePower0 > 350 and horsePower0 < 500 and weight0 > 1000 and weight0 < 1400 and
      manufacturer <> nil and model <> ""
  post model = model0 and manufacturer = manufacturer0 and weight = weight0 and horsePower =
      horsePower0;

 /**
  * Gets the car model.
  *
  * @return model
  */

 pure public GetModel: () ==> String
  GetModel() ==
    return model
  post RESULT = model;

 /**
  * Gets the car horse power.
  *
  * @return horsePower
  */

 pure public GetHorsePower: () ==> nat
  GetHorsePower() ==
    return horsePower
  post RESULT = horsePower;

 /**
  * Gets the car weight.
  *
  * @return weight
  */

 pure public GetWeight: () ==> nat
  GetWeight() ==
    return weight
  post RESULT = weight;

 /**
  * Gets the manufacturer, if it exists.
  *
  * @return Manufacturer
  */

 pure public GetManufacturer: () ==> [Manufacturer]
  GetManufacturer() ==
    return manufacturer
  post RESULT = manufacturer;

end Car
```

3.2 Championship

```
class Championship is subclass of State

types
 public String = seq of char;
 public Date = DateUtils`Date;
  public TeamName = Team`TeamName;

instance variables
  private series: seq of Rally := [];
  private currentRallyIndex: int := -1;
  private teams: map TeamName to Team := { |-> };
  private partners: set of Partner := {};
  private name: String;
  private startDate: Date;
  private endDate: Date;

  inv not exists r1, r2 in seq series &
        r1 <> r2 and r1.GetName() = r2.GetName();
  inv not exists p1, p2 in set partners &
        p1 <> p2 and p1.GetName() = p2.GetName();
 inv endDate > startDate;
  inv forall i in set inds series &
      i > 1 => let s1 = series(i-1), s2 = series(i) in s1.GetStartDate() < s2.GetStartDate();
 inv forall r in seq series & r.GetEndDate() < endDate and r.GetStartDate() > startDate;

operations
 /**
  * Instantiates a championship instance.
  */

  public Championship: String * Date * Date ==> Championship
  Championship(name0, startDate0, endDate0) == (
     name := name0;
     startDate := startDate0;
     endDate := endDate0;
     return self;
   )
  pre endDate0 > startDate0 and name <> ""
  post name = name0 and startDate = startDate0 and endDate = endDate0 and teams = { |-> };

 -- *** Transactions ***

 /**
  * Starts the championship, initializing the first rally participants
  */

 public StartChampionship: () ==> ()
  StartChampionship() == (
   currentRallyIndex := 1;
    series(currentRallyIndex).StartRally(GetDrivers());
   Init();
  )
  pre state = <OffSeason> and series <> [] and teams <> { |-> } and not exists s in seq series
       & s.GetState() <> <OffSeason>
  post state = <Occurring> and currentRallyIndex <> -1 and series(currentRallyIndex).GetState
       () = <Occurring>;

 /**
  * Updates the current rally.
  * The set of performances is used to define the driver performances of the current stage
      that is happening
  */

 public UpdateCurrentRally: set of Performance ==> ()
  UpdateCurrentRally(performances) == (
   series(currentRallyIndex).NextStage(performances);
  )
  pre state = <Occurring> and series(currentRallyIndex).GetState() = <Occurring>;

 /**
  * Checks if the current rally is completed
  *
  * @return bool
  */

 public IsCurrentRallyCompleted: () ==> bool
  IsCurrentRallyCompleted() == (
   if(series(currentRallyIndex).GetState() = <Completed>)
    then return true
   else return false
  )
  pre state = <Occurring>;

 /**
  * Starts the next rally, only if the current one is completed
  */

 public NextRally: () ==> ()
  NextRally() == (
   currentRallyIndex := currentRallyIndex + 1;
    if(currentRallyIndex > len series) then EndChampionship() else series(currentRallyIndex).
        StartRally(GetDrivers());
  )
  pre state = <Occurring> and series(currentRallyIndex).GetState() = <Completed>;

 /**
  * Ends the championship
  */

 private EndChampionship: () ==> ()
  EndChampionship() == (
   currentRallyIndex := -1;
   Finalize();
  )
  pre state = <Occurring>
  post state = <Completed> and not exists r in seq series & r.GetState() <> <Completed>;

 /**
  * Adds a partner to the championship
  */

 public AddPartner: Partner ==> ()
  AddPartner(p) == (
   partners := partners union {p}
  )
  pre state = <OffSeason> and not exists p1 in set partners & p1.GetName() = p.GetName()
  post p in set partners;

 /**
  * Gets the points of a team on a specific rally
  *
  * @return int
  */

 public GetTeamRallyRanking: String * String ==> int
```

```
 GetTeamRallyRanking(teamName, rallyName) == (
  let i in set inds series be st series(i).GetName() = rallyName in return series(i).
      GetTeamRanking(teamName);
 )
 pre teamName in set dom teams and exists r in seq series & r.GetName() = rallyName;

/**
 * Gets the global ranking of a single driver, during the respective championship
 *
 * @return int
 */

 pure public GetDriverGlobalRanking: Driver ==> int
  GetDriverGlobalRanking(d) == (
   dcl points: int := 0;
   for rally in series
    do
     if(rally.GetState() <> <OffSeason>)
      then (points := points + rally.GetDriverRanking(d).GetPoints(););
  return points;
  );

/**
 * Gets the global ranking (points) of a team
 *
 * @return int
 */

public GetTeamGlobalRanking: String ==> int
 GetTeamGlobalRanking(teamName) == (
  dcl points: int := 0;
   for elem in series
     do
       if (elem.GetState() <> <OffSeason>)
           then (points := points + elem.GetTeamRanking(teamName););

 return points;
 )
 pre teamName in set dom teams;

/**
 * Gets the championship global ranking, with the points of each team
 *
 * @return map String to int
 */

public GetTeamsRanking: () ==> map String to int
 GetTeamsRanking() == (
   dcl rankings: map String to int := { |-> };
   for all elem in set dom teams
    do
     rankings := rankings munion { elem |-> GetTeamGlobalRanking(elem) };
   return rankings;
 );

/**
 * Removes a partner from the championship
 */

public RemovePartner: Partner ==> ()
 RemovePartner(p) == (
  partners := partners \ {p}
 )
 pre state = <OffSeason> and p in set partners
 post partners = partners~ \ {p};

/**
 * Adds a rally to the championship, whose dates must be between the championship start and
     end
 * Furthermore, the rally state must be OffSeason
 */
```

```
public AddRally: Rally ==> ()
 AddRally(r) == (
  series := series ^ [r]
 )
 pre r.GetStages() <> [] and r.GetStartDate() > startDate and r.GetEndDate() < endDate and
     state = <OffSeason> and not exists r1 in seq series & r1.GetName() = r.GetName();

/**
 * Adds a team to the championship
 */

public AddTeam: Team ==> ()
 AddTeam(t) == (
  teams := teams munion {t.GetName() |-> t};
 )
 pre state = <OffSeason> and t.GetName() not in set dom teams
 post teams = teams~ munion {t.GetName() |-> t};

/**
 * Removes a team from the championship
 */

public RemoveTeam: Team ==> ()
 RemoveTeam(t) == (
  teams := {t.GetName()} <-: teams;
 )
 pre state = <OffSeason> and t.GetName() in set dom teams
 post t.GetName() not in set dom teams;

-- *** Getters ***

/**
 * Gets the current rally that its occurring
 *
 * @return Rally
 */

pure public GetCurrentRally: () ==> [Rally]
 GetCurrentRally() == (
  return
   if state = <Occurring> and currentRallyIndex >= 1
          then series(currentRallyIndex)
       else nil;
 );

/**
 * Gets the championship set of drivers
 *
 * @return set of Driver
 */

pure public GetDrivers: () ==> set of Driver
 GetDrivers() ==
  return dunion { t.GetDrivers() | t in set rng teams };

/**
 * Gets the championship name.
 *
 * @return String
 */

pure public GetName: () ==> String
 GetName() ==
   return name
 post RESULT = name;

/**
 * Gets the championship start date.
 *
 * @return Date
 */

pure public GetStartDate: () ==> Date
 GetStartDate() ==
   return startDate
 post RESULT = startDate;

/**
 * Gets the championship end date.
 *
 * @return Date
 */

pure public GetEndDate: () ==> Date
 GetEndDate() ==
   return endDate
 post RESULT = endDate;

/**
 * Gets the championship teams
 *
 * @return map TeamName to Team
 */

pure public GetTeams: () ==> map TeamName to Team
GetTeams() ==
 return teams
post RESULT = teams;

/**
 * Gets the championship partners
 *
 * @return set of Partner
 */

pure public GetPartners: () ==> set of Partner
 GetPartners() ==
  return partners
 post RESULT = partners;

/**
 * Gets the sequence of rallies
 *
 * @return seq of Rally
 */

pure public GetSeries: () ==> seq of Rally
 GetSeries() ==
  return series
 post RESULT = series;

/**
 * Gets a team that matches the name given
 *
 * @return Team
 */

pure public GetTeamByName: (String) ==> Team
 GetTeamByName(tName) ==
  return teams(tName)
 pre tName in set dom teams;

/**
 * Gets a rally that matches the name given
 *
 * @return Rally
 */

pure public GetRallyByName: (String) ==> Rally
 GetRallyByName(rName) ==
  let i in set inds series be st series(i).GetName()=rName in return series(i)
  pre exists i in set inds series & series(i).GetName() = rName;

end Championship
```

3.3 DateUtils

```
class DateUtils

  types
    public Date = nat
       inv d == IsValidDate(d div 10000, (d div 100) mod 100, d mod 100);

  values
    public MinDate = MakeDate(1,1,1);

  functions


  public static IsValidDate: nat * nat * nat -> bool
   IsValidDate(year, month, day) ==
    year >= 1 and month >= 1 and month <= 12 and day >= 1 and day <= DaysOfMonth(year, month);


  public static IsLeapYear: nat -> bool
   IsLeapYear(year) ==
    year mod 4 = 0 and year mod 100 <> 0 or year mod 400 = 0;


    public static DaysOfMonth: nat * nat -> nat
     DaysOfMonth(year, month) == (
      cases month :
         1, 3, 5, 7, 8, 10, 12 -> 31,
         4, 6, 9, 11 -> 30,
         2 -> if IsLeapYear(year) then 29 else 28
       end
      )
     pre month >= 1 and month <= 12;


  public static MakeDate: nat * nat * nat -> Date
   MakeDate(year, month, day) ==
    year * 10000 + month * 100 + day
   pre IsValidDate(year, month, day);


   public static Year: Date -> nat
    Year(d) ==
     d div 10000;


   public static Month: Date -> nat
    Month(d) ==
     (d div 100) mod 100;


   public static Day: Date -> nat
    Day(d) ==
     d mod 100;

end DateUtils
```

3.4 Driver

```
class Driver

types
 public String = seq of char;
 public Country = Team`Country;
 public Date = DateUtils`Date;
 public Gender = <Male> | <Female>;

instance variables
  private team: [Team] := nil;
  private car: [Car] := nil;
  private name: String;
  private gender: Gender;
  private nationality: Country;
  private birth: Date;
  private true_self: Driver;

  inv if(team <> nil and car <> nil and team.GetBacker() <> nil) then team.GetBacker() = car.
      GetManufacturer() else true;
  -- Assured by the inv in each Team
  -- inv if(team <> nil) then true_self in set team.GetDrivers() else true;

operations

 /**
  * Instantiates a driver instance.
  */

  public Driver: String * Gender * Country * Date ==> Driver
  Driver(name0, gender0, nationality0, birth0) == (
    name := name0;
    gender := gender0;
    nationality := nationality0;
    birth := birth0;
    true_self := self;
    return self;
  )
  pre name0 <> ""
  post name = name0 and gender = gender0 and nationality = nationality0 and birth = birth0 and
       team = nil and car = nil;

 -- *** Transactions **

 /**
  * Sets the driver team.
  */

 public SetTeam: (Team) ==> ()
  SetTeam(t) ==
    team := t
  pre car = nil
  post team = t;

 /**
  * Removes the current team
  */

 public RemoveTeam: () ==> ()
  RemoveTeam() ==
    team := nil
  post team = nil;

 /**
  * Removes the current car
  */

 public RemoveCar: () ==> ()
  RemoveCar() ==
    car := nil
  post car = nil;
```

```
/**
 * Sets the driver car
 */

public SetCar: (Car) ==> ()
 SetCar(c) ==
   car := c
 pre team <> nil and c in set team.GetCars()
 post car = c;

-- *** Getters ***

/**
 * Gets the driver current team.
 *
 * @return team
 */

pure public GetTeam: () ==> [Team]
 GetTeam() ==
   return team
 post RESULT = team;

/**
 * Gets the driver car.
 *
 * @return Car
 */

pure public GetCar: () ==> [Car]
 GetCar() ==
   return car
 post RESULT = car;

/**
 * Gets the driver name.
 *
 * @return name
 */

pure public GetName: () ==> String
 GetName() ==
   return name
 post RESULT = name;

/**
 * Gets the driver gender.
 *
 * @return gender
 */

pure public GetGender: () ==> Gender
 GetGender() ==
   return gender
 post RESULT = gender;

/**
 * Gets the driver nationality.
 *
 * @return country
 */

pure public GetNationality: () ==> Country
 GetNationality() ==
   return nationality
 post RESULT = nationality;

/**
 * Gets the birth date.
 *
 * @return Date
```

```
  */

 pure public GetBirth: () ==> Date
  GetBirth() ==
    return birth
  post RESULT = birth;

end Driver
```

3.5 Manufacturer

```
class Manufacturer

types
 public String = seq of char;
 public Country = Team`Country;

instance variables
  private name: String;
  private description: String;
  private country: Country;

operations
 /**
  * Instantiates a manufacturer instance.
  */

  public Manufacturer : String * String * Country ==> Manufacturer
  Manufacturer(name0, description0, country0) == (
    name := name0;
    description := description0;
    country := country0;
    return self;
  )
  pre name0 <> "" and description0 <> ""
  post name = name0 and description = description0 and country = country0;

 /**
  * Gets the manufacturer name.
  *
  * @return name
  */

 pure public GetName: () ==> String
  GetName() ==
    return name
  post RESULT = name;

 /**
  * Gets the manufacturer description.
  *
  * @return description
  */

 pure public GetDescription: () ==> String
  GetDescription() ==
    return description
  post RESULT = description;

 /**
  * Gets the manufacturer country.
  *
  * @return country
  */

 pure public GetCountry: () ==> Country
  GetCountry() ==
    return country
  post RESULT = country;

end Manufacturer
```

3.6 Partner

```
class Partner

types
 public String = seq of char;

instance variables
  private name: String;
  private description: String;

operations
 /**
  * Instantiates a partner instance.
  */

  public Partner : String * String ==> Partner
  Partner(name0, description0) == (
    name := name0;
    description := description0;
    return self;
  )
  pre name0 <> "" and description0 <> ""
  post name = name0 and description = description0;

 /**
  * Gets the partner name.
  *
  * @return name
  */

 pure public GetName: () ==> String
  GetName() ==
    return name
  post RESULT = name;

 /**
  * Gets the partner description.
  *
  * @return description
  */

 pure public GetDescription: () ==> String
  GetDescription() ==
    return description
  post RESULT = description;

end Partner
```

3.7 Performance

```
class Performance

instance variables
  private driver: [Driver] := nil;
  private averageSpeed: real := 0.0;
  private time: real := 0.0;
  private penalty: nat := 0;
```

```
 inv driver.GetTeam() <> nil and driver.GetCar() <> nil;

operations
 /**
  * Instantiates a performance instance.
  */

  public Performance : Driver ==> Performance
  Performance(driver0) == (
   driver := driver0;
    return self;
  )
  post driver = driver0;

 /**
  * Gets the performance driver.
  *
  * @return Driver
  */

 pure public GetDriver: () ==> [Driver]
  GetDriver() ==
    return driver
  post RESULT = driver;

 /**
  * Gets the performance average speed.
  *
  * @return real
  */

 pure public GetAverageSpeed: () ==> real
  GetAverageSpeed() ==
    return averageSpeed
  post RESULT = averageSpeed;

 /**
  * Sets the performance average speed.
  */

 public SetAverageSpeed: real ==> ()
  SetAverageSpeed(avg) ==
    averageSpeed := avg
  post avg = averageSpeed;

 /**
  * Gets the performance time.
  *
  * @return real
  */

 pure public GetTime: () ==> real
  GetTime() ==
    return time
  post RESULT = time;

 /**
  * Sets the performance time.
  */

 public SetTime: real ==> ()
  SetTime(t) ==
    time := t
  post t = time;

 /**
  * Gets the performance penalty.
  *
  * @return nat
  */
 pure public GetPenalty: () ==> nat
  GetPenalty() ==
    return penalty
  post RESULT = penalty;

 /**
  * Sets the performance penalty.
  */

 public SetPenalty: nat ==> ()
  SetPenalty(p) ==
    penalty := p
  post p = penalty;

end Performance
```

3.8 Rally

```
class Rally is subclass of State

types
 public Country = Team`Country;
 public Surface = <Mountain> | <Asphalt> | <Sand> | <Forest> | <Snow> | <Gravel>;
 public String = seq of char;
 public Date = DateUtils`Date;
 public StageId = Stage`StageId;

instance variables
  private stages: seq of Stage := [];
  private currentStageIndex: int := -1;
  private rankings: seq of Ranking := [];
  private startDate: Date;
  private endDate: Date;
  private name: String;
  private description: String;
  private country: Country;
  private surface: Surface;
  private distance: real := 0.0;

  inv endDate > startDate;
  inv not exists s1, s2 in seq stages & s1 <> s2 and s1.GetId() = s2.GetId();
  inv forall i in set inds stages &
      i > 1 => let s1 = stages(i-1), s2 = stages(i) in s1.GetId() < s2.GetId();
  inv not exists r1, r2 in seq rankings &
      r1 <> r2 and r1.GetDriver().GetName() = r2.GetDriver().GetName();
  -- As a post condition
  -- inv forall i in set inds rankings &
      -- i > 1 => let e1 = rankings(i-1), e2 = rankings(i) in e1.GetPoints() >= e2.GetPoints()
          ;

operations
 /**
  * Instantiates a rally instance.
  */

  public Rally : String * String * Country * Surface * Date * Date ==> Rally
  Rally(name0, description0, country0, surface0, startDate0, endDate0) == (
   name := name0;
   description := description0;
   country := country0;
   surface := surface0;
   startDate := startDate0;
   endDate := endDate0;
    return self;
  )
  pre endDate0 > startDate0 and name0 <> "" and description0 <> ""
  post name = name0 and description = description0 and country = country0
   and surface = surface0 and startDate = startDate0
   and endDate = endDate0 and rankings = [] and stages = [];

 -- *** Transactions ***

 /**
  * Starts a rally by changing the rally state, and initializing the ranking of the drivers
      that will participate
  */

 public StartRally: set of Driver ==> ()
  StartRally(drivers) == (
   for all elem in set drivers
     do rankings := rankings ^ [new Ranking(elem)];
    currentStageIndex := 1;
    stages(currentStageIndex).Init();
   Init();
  )
  pre state = <OffSeason> and stages <> [] and drivers <> {} and not exists s in seq stages &
      s.GetState() <> <OffSeason>
  post state = <Occurring> and rankings <> [];

 /**
  * Receives a set of performances, and terminates the current stage that is happening.
  * The performances are added to the respective stage and then rally ranking points are
      updated accordingly
  * If the stage is the last one, it ends the rally
  * As a post condition, all the rankings must be properly ordered
  */

 public NextStage: set of Performance ==> ()
  NextStage(performances) == (
   stages(currentStageIndex).Finalize();
   for all elem in set performances
     do (
      stages(currentStageIndex).AddPerformance(elem);
      let i in set inds rankings be st elem.GetDriver() = rankings(i).GetDriver() in
         rankings(i).AddPoints(elem.GetTime());
      );
    rankings := BubbleSort(rankings);

    currentStageIndex := currentStageIndex + 1;
    if(currentStageIndex > len stages) then EndRally() else stages(currentStageIndex).Init();
  )
  pre state = <Occurring> and stages(currentStageIndex).GetState() = <Occurring>
  post forall i in set inds rankings &
      i > 1 => let e1 = rankings(i-1), e2 = rankings(i) in e1.GetPoints() >= e2.GetPoints();

 /**
  * Ends the rally
  */

 private EndRally: () ==> ()
  EndRally() == (
   currentStageIndex := -1;
   Finalize();
  )
  pre state = <Occurring>
  post state = <Completed> and not exists s in seq stages & s.GetState() <> <Completed>;

 /**
  * Returns the sum of points from the drivers that belong to a team, that is, the team points
  *
  * @return int
  */

 public GetTeamRanking: String ==> int
  GetTeamRanking(teamName) == (
    dcl points: int := 0;
     for elem in rankings
```

```
      do
        if (elem.GetDriver().GetTeam().GetName() = teamName)
            then (points := points + elem.GetPoints(););

  return points;
 )
 pre state <> <OffSeason>;

/**
 * Adds a new stage, whose dates must be between the rally start and end
 */

public AddStage: Stage ==> ()
 AddStage(s) == (
  stages := stages ^ [s];
  distance := distance + s.GetDistance();
 )
 pre state = <OffSeason> and s.GetDate() >= startDate and s.GetDate() <= endDate and not
     exists s1 in seq stages & s <> s1 and s1.GetId() = s.GetId()
 post exists s1 in seq stages & s1 = s;

-- *** Getters ***

/**
 * Gets the rally starting date.
 *
 * @return Date
 */

pure public GetStartDate: () ==> Date
 GetStartDate() ==
   return startDate
 post RESULT = startDate;

/**
 * Gets the rally end date.
 *
 * @return Date
 */

pure public GetEndDate: () ==> Date
 GetEndDate() ==
   return endDate
 post RESULT = endDate;

/**
 * Gets the rally name.
 *
 * @return String
 */

pure public GetName: () ==> String
 GetName() ==
   return name
 post RESULT = name;

/**
 * Gets the rally description.
 *
 * @return String
 */

pure public GetDescription: () ==> String
 GetDescription() ==
   return description
 post RESULT = description;

/**
 * Gets the rally Country.
 *
 * @return Country
 */

pure public GetCountry: () ==> Country
 GetCountry() ==
   return country
 post RESULT = country;

/**
 * Gets the rally Surface.
 *
 * @return Surface
 */

pure public GetSurface: () ==> Surface
 GetSurface() ==
   return surface
 post RESULT = surface;

/**
 * Gets the rally total distance.
 *
 * @return real
 */

pure public GetDistance: () ==> real
 GetDistance() ==
   return distance
 post RESULT = distance;

/**
 * Gets the rally stages.
 *
 * @return seq of Stage
 */

pure public GetStages: () ==> seq of Stage
 GetStages() ==
  return stages
 post RESULT = stages;

/**
 * Gets the rally rankings.
 *
 * @return seq of Ranking
 */

pure public GetRankings: () ==> seq of Ranking
 GetRankings() ==
  return rankings
 post RESULT = rankings;

/**
 * Gets the rally current stage.
 *
 * @return Stage
 */

pure public GetCurrentStage: () ==> Stage
 GetCurrentStage() ==
  return stages(currentStageIndex)
 pre currentStageIndex >= 1
 post exists stage in seq stages & stage = RESULT;

/**
 * Gets the specified driver ranking
 *
 * @return Ranking
 */

pure public GetDriverRanking: Driver ==> Ranking
 GetDriverRanking(driver0) ==
   let i in set inds rankings be st rankings(i).GetDriver() = driver0 in return rankings(i)
  pre exists ranking in seq rankings & ranking.GetDriver() = driver0;

 /**
  * Gets the specified driver performance on the defined stage.
  *
  * @return Performance
  */

 pure public GetDriverPerformanceOnStage: StageId * Driver ==> Performance
    GetDriverPerformanceOnStage(stageId0, driver0) ==
     let iS in set inds stages be st stages(iS).GetId() = stageId0 in
        (let iP in set inds stages(iS).GetPerformances() be st stages(iS).GetPerformances()(iP
            ).GetDriver() = driver0 in
           return stages(iS).GetPerformances()(iP))
  pre exists stage in seq stages & stage.GetId()=stageId0 and
     (exists performance in seq stage.GetPerformances() & performance.GetDriver() = driver0);

 /**
  * Orders a sequence of rankings, using the algorithm bubble sort
  *
  * @return seq of Ranking
  */

 private static BubbleSort : seq of Ranking ==> seq of Ranking
  BubbleSort (k) == (
   dcl sorted_list : seq of Ranking := k;
   for i = len k to 1 by -1 do
    for j = 1 to i-1 do
     if sorted_list(j).GetPoints() < sorted_list(j+1).GetPoints()
      then (dcl temp: Ranking := sorted_list(j);
     sorted_list(j) := sorted_list(j+1);
     sorted_list(j+1) := temp
   );
   return sorted_list
  )

end Rally
```

3.9 Ranking

```
class Ranking

instance variables
  private driver: [Driver] := nil;
  private points: int := 0;

  inv driver.GetTeam() <> nil and driver.GetCar() <> nil;

operations
 /**
  * Instantiates a ranking instance.
  */

  public Ranking: Driver ==> Ranking
  Ranking(driver0) == (
   driver := driver0;
    return self;
  )
  post driver = driver0 and points = 0;

 /**
  * Adds points to the ranking instance
  */

 public AddPoints: int ==> ()
  AddPoints(score) ==
   points := points + score;

 -- *** Getters ***

 /**
  * Gets the instance driver.
  *
  * @return Driver
  */

 pure public GetDriver: () ==> [Driver]
  GetDriver() ==
    return driver
  post RESULT = driver;

 /**
  * Gets the number of points made by the driver, so far.
  *
  * @return int
  */

 pure public GetPoints: () ==> int
  GetPoints() ==
    return points
  post RESULT = points;

end Ranking
```

3.10 Sponsor

```
class Sponsor

types
 public String = seq of char;

instance variables
  private name: String;
  private description: String;

operations
 /**
  * Instantiates a sponsor instance.
  */

  public Sponsor : String * String ==> Sponsor
  Sponsor(name0, description0) == (
    name := name0;
    description := description0;
    return self;
  )
  pre name0 <> "" and description0 <> ""
  post name = name0 and description = description0;

 /**
  * Gets the sponsor name.
  *
  * @return name
  */

 pure public GetName: () ==> String
  GetName() ==
    return name
  post RESULT = name;

 /**
  * Gets the sponsor description.
  *
  * @return description
  */

 pure public GetDescription: () ==> String
  GetDescription() ==
    return description
  post RESULT = description;

end Sponsor
```

3.11 Stage

```
class Stage is subclass of State

types
 public StageType = <Transport> | <Special>;
 public Date = DateUtils`Date;
 public StageId = nat1;
 public String = seq of char;

instance variables
  private performances: seq of Performance := [];
  private type: StageType;
  private date: Date;
  private distance: real;
  private id: StageId;

  -- performances have unique drivers
  inv not exists p1, p2 in seq performances &
        p1<>p2 and p1.GetDriver() = p2.GetDriver();
 inv if(state = <OffSeason> or state = <Occurring>) then len performances = 0 else forall p in
      seq performances & p.GetTime() > 0;

operations
 /**
  * Instantiates a stage instance.
  */

  public Stage : nat1 * StageType * Date * real ==> Stage
  Stage(id0, type0, date0, distance0) == (
   id := id0;
   type := type0;
   date := date0;
   distance := distance0;
    return self;
  )
  pre distance0 > 0 and id0 <> 0
  post type = type0 and date = date0 and distance = distance0 and id = id0 and state = <
      OffSeason>;

 -- *** Transactions ***

 /**
  * Adds a new performance to the stage. Performances are unique and a stage can have the
      performances of all drivers that participated in the rally or not.
  * A performance can be added to the stage performances only if the stage is completed
  */

 public AddPerformance: Performance ==> ()
  AddPerformance(perf) ==
   performances := InsertSorted(perf, performances, Compare)
  pre state = <Completed>
  post exists p in seq performances & perf = p;

 /**
  * Finds the driver with the specified name and returns the performance, if it exists
  *
  * @return Performance
  */

 public GetDriverPerformance: String ==> [Performance]
  GetDriverPerformance(driverName) == (
   return
    if exists p in seq performances & p.GetDriver().GetName() = driverName
           then iota p in seq performances & p.GetDriver().GetName() = driverName
        else nil;
  );

 /**
  * Gets the specified driver performance, if it exists
  *
  * @return Performance
  */
 public GetDriverPerformance: Driver ==> [Performance]
  GetDriverPerformance(d) == (
   return
    if exists p in seq performances & p.GetDriver() = d
           then iota p in seq performances & p.GetDriver() = d
        else nil;
  );

 -- *** Getters ***

 /**
  * Gets the stage type.
  *
  * @return StageType
  */

 pure public GetType: () ==> StageType
  GetType() ==
    return type
  post RESULT = type;

 /**
  * Gets the stage identifier.
  *
  * @return StageId
  */

 pure public GetId: () ==> StageId
  GetId() ==
    return id
  post RESULT = id;

 /**
  * Gets the stage date.
  *
  * @return Date
  */

 pure public GetDate: () ==> Date
  GetDate() ==
    return date
  post RESULT = date;

 /**
  * Gets the stage total distance.
  *
  * @return real
  */

 pure public GetDistance: () ==> real
  GetDistance() ==
    return distance
  post RESULT = distance;

 /**
```

```
  * Gets the stage performances
  *
  * @return seq of performances
  */

 pure public GetPerformances: () ==> seq of Performance
  GetPerformances() ==
   return performances
  post RESULT = performances;

functions

 private static InsertSorted: Performance * seq of Performance * (Performance * Performance ->
      int) -> seq of Performance
  InsertSorted(i, l, compare) ==
   if(l = []) then [i]
   else if (compare(i, hd l) <= 0) then [i] ^ l
   else [hd l] ^ InsertSorted(i, tl l, compare);


 private static Compare: Performance * Performance -> int
  Compare(p1, p2) ==
   if p1.GetTime() > p2.GetTime() then 1 else -1
  pre p1.GetTime() > 0 and p2.GetTime() > 0;

end Stage
```

3.12 State

```
class State

types
 public State = <OffSeason> | <Completed> | <Occurring>;

instance variables
 protected state: State := <OffSeason>;

operations

 public Init: () ==> ()
  Init() ==
   state := <Occurring>
   pre state = <OffSeason>
   post state = <Occurring>;


 public Finalize: () ==> ()
  Finalize() ==
   state := <Completed>
   pre state = <Occurring>
   post state = <Completed>;


 pure public GetState: () ==> State
  GetState() ==
    return state
  post RESULT = state;

end State
```

3.13 Team

```
class Team
```

```
types
 public Country = <Italy> | <France> | <Spain> | <Portugal> |
          <SouthKorea> | <Germany> | <GreatBritain> | <Japan>;
 public String = seq of char;
 public TeamName = seq of char;

instance variables
  private drivers : set of Driver := {};
  private sponsors : set of Sponsor := {};
  private backer : [Manufacturer] := nil;
  private cars : set of Car := {};
  private name : String;
  private based : Country;
  private true_self: Team;

  inv not exists d1, d2 in set drivers &
        d1 <> d2 and d1.GetName() = d2.GetName();
  inv not exists s1, s2 in set sponsors &
        s1 <> s2 and s1.GetName() = s2.GetName();
  inv not exists c1, c2 in set cars &
        c1 <> c2 and c1.GetModel() = c2.GetModel();

  inv forall d in set drivers & d.GetTeam() = true_self;
  inv if(backer <> nil) then forall c in set cars & c.GetManufacturer() = backer else true;
  inv forall d in set drivers & d.GetCar() <> nil => d.GetCar() in set cars;

operations
 /**
  * Instantiates a team instance.
  */

  public Team : String * Country ==> Team
  Team(name0, based0) == (
   name := name0;
   based := based0;
   true_self := self;
    return self;
  )
  pre name0 <> ""
  post name = name0 and based = based0;

 -- *** Transactions ***

 /**
  * Adds a new sponsor
  */

 public AddSponsor: Sponsor ==> ()
  AddSponsor(s) == (
   sponsors := sponsors union {s}
  )
  pre not exists s1 in set sponsors & s1.GetName() = s.GetName();

 /**
  * Removes a sponsor that must exist
  */

 public RemoveSponsor: Sponsor ==> ()
  RemoveSponsor(s) == (
   sponsors := sponsors \ {s}
  )
  pre s in set sponsors
  post sponsors = sponsors~ \ {s};

 /**
  * Sets the team manufacturer
  */

 public SetBacker: Manufacturer ==> ()
  SetBacker(m) == (
   backer := m;
  )
  pre not exists c1 in set cars & c1.GetManufacturer().GetName() <> m.GetName();

 /**
  * Removes the team manufacturer
  */

 public RemoveBacker: () ==> ()
  RemoveBacker() == (
   backer := nil;
  )
  post backer = nil;

 /**
  * Adds a new car
  */

 public AddCar: Car ==> ()
  AddCar(c) == (
   cars := cars union {c}
  )
  pre if(backer <> nil) then c.GetManufacturer().GetName() = backer.GetName() else true
   and not exists c1 in set cars & c1.GetModel() = c.GetModel()
  post c in set cars;

 /**
  * Removes a car, that must exist
  */

 public RemoveCar: Car ==> ()
  RemoveCar(c) == (
   cars := cars \ {c}
  )
  pre c in set cars and forall d in set drivers & d.GetCar() <> c
  post cars = cars~ \ {c};

 /**
  * Assigns a car to a driver. Both must belong in the respective team
  */

 public AssignCarToDriver: Car * Driver ==> ()
  AssignCarToDriver(car, driver) == (
      for all elem in set drivers
       do
         if elem.GetName() = driver.GetName()
            then (elem.SetCar(car))
    )
    pre car in set cars and driver in set drivers and driver.GetCar() = nil;

 /**
  * Removes a car assignment
  */

 public UnassignCarFromDriver: Driver ==> ()
  UnassignCarFromDriver(driver) == (
      for all elem in set drivers
       do
         if elem.GetName() = driver.GetName()
            then (elem.RemoveCar())
    )
    pre driver in set drivers and driver.GetCar() <> nil;

 /**
  * Adds a new driver
  */

 public AddDriver: Driver ==> ()
  AddDriver(d) == (
   d.SetTeam(self);
   drivers := drivers union {d}
  )
  pre d.GetTeam() = nil and d.GetCar() = nil and not exists d1 in set drivers & d1.GetName() =
       d.GetName();

 /**
  * Removes a driver
  */

 public RemoveDriver: Driver ==> ()
  RemoveDriver(d) == (
   drivers := drivers \ {d};
   d.RemoveTeam();
   d.RemoveCar();
  )
  pre d in set drivers
  post drivers = drivers~ \ {d};

 -- *** Getters ***

 /**
  * Gets the team cars
  *
  * @return set of Car
  */

 pure public GetCars: () ==> set of Car
  GetCars() ==
    return cars
  post RESULT = cars;

 /**
  * Gets the team sponsors.
  *
  * @return set of sponsor
  */

 pure public GetSponsors: () ==> set of Sponsor
  GetSponsors() ==
    return sponsors
  post RESULT = sponsors;

 /**
  * Gets the team drivers
  *
  * @return set of Driver
  */

 pure public GetDrivers: () ==> set of Driver
  GetDrivers() ==
    return drivers
  post RESULT = drivers;

 /**
  * Gets the team manufacturer
  *
  * @return Manufacturer
  */

 pure public GetBacker: () ==> [Manufacturer]
  GetBacker() ==
    return backer
  post RESULT = backer;

 /**
  * Gets the team name.
  *
  * @return name
  */

 pure public GetName: () ==> String
  GetName() ==
    return name
  post RESULT = name;

 /**
  * Gets the team original country.
  *
  * @return Country
  */

 pure public GetBased: () ==> Country
  GetBased() ==
    return based
  post RESULT = based;

 /**
  * Gets a driver instance, through its name.
  *
  * @return Driver
  */

 pure public GetDriverByName: (String) ==> Driver
  GetDriverByName(dName) ==
   let driver in set drivers be st driver.GetName()=dName in return driver
  pre let driver in set drivers in driver.GetName() = dName;

end Team
```

4 Model validation

4.1 CarTest

```
class CarTest is subclass of MyTestCase

instance variables
 manufacturer1: Manufacturer := new Manufacturer("Citroen",
  "Citroen is one of the world's leading mainstream car manufacturers.",
  <France>);
 manufacturer2: Manufacturer := new Manufacturer("Toyota",
  "Toyota is one of the worlds best-known and most successful businesses, building cars and
      trucks in 26 countries for sale in more than 160 markets around the globe.",
  <Japan>);
 manufacturer3: Manufacturer := new Manufacturer("Hyunday",
  "Hyundai Motor Company leads the Hyundai Motor Group, a wide-reaching business capable of
      processing resources from molten iron to finished cars.",
  <SouthKorea>);

 car1: Car := new Car("Hyundai i20", manufacturer3, 370, 1250);
 car2: Car := new Car("Citroen C3", manufacturer1, 400, 1300);
 car3: Car := new Car("Toyota Yaris", manufacturer2, 480, 1200);

operations

 public Run: () ==> ()
 Run() == (
  IO`println("\nCar Tests");

  assertEqual("Hyundai i20", car1.GetModel());
  assertEqual(manufacturer1, car2.GetManufacturer());
  assertEqual(480, car3.GetHorsePower());
  assertEqual(1300, car2.GetWeight());

  IO`println("\nFinalizing Car Tests");
 );

end CarTest
```

Function or operation	Line	Coverage	Calls
Car	16	100.0%	28
GetHorsePower	42	100.0%	1
GetManufacturer	62	100.0%	5
GetModel	32	100.0%	321
GetWeight	52	100.0%	1
Car.vdmpp		100.0%	356

4.2 ChampionshipTest

```
class ChampionshipTest is subclass of MyTestCase

instance variables
 champ1: Championship := new Championship("WRC", DateUtils`MakeDate(2017, 10, 2), DateUtils`
     MakeDate(2017, 11, 20));
 champ2: Championship := new Championship("WRC Junior", DateUtils`MakeDate(2017, 8, 23),
     DateUtils`MakeDate(2017, 10, 11));
 champ3: Championship := new Championship("WRC Pro", DateUtils`MakeDate(2017, 1, 13),
     DateUtils`MakeDate(2017, 3, 20));
 champ4: Championship := new Championship("Amateur WRC", DateUtils`MakeDate(2017, 2, 2),
     DateUtils`MakeDate(2017, 5, 15));
 sponsor1: Sponsor := new Sponsor("Abu Dhabi", "Official visitor website for Abu Dhabi travel
     and tourism, offering information on hotels, restaurants, things to do, culture &
     heritage and events.");
 sponsor2: Sponsor := new Sponsor("Eparco", "Specialiste assainissement compact, developpe et
     fabrique en exclusivite des solutions pour lassainissement non collectif depuis 45 ans.")
     ;
 sponsor3: Sponsor := new Sponsor("Stilo", "Stilo helmets competition. Top functionality,
     everything a driver needs must be standard feature: intercom, connections, earmuffs,
     drinking system.");
 sponsor4: Sponsor := new Sponsor("Michelin", "Michelin, the leading tire company, is
     dedicated to enhancing its clients mobility, sustainably; designing and distributing the
     most suitable tires, services and solutions for its clients needs.");

 manufacturer1: Manufacturer := new Manufacturer("Citroen",
  "Citroen is one of the worlds leading mainstream car manufacturers.",
  <France>);
 manufacturer2: Manufacturer := new Manufacturer("Toyota",
  "Toyota is one of the worlds best-known and most successful businesses, building cars and
      trucks in 26 countries for sale in more than 160 markets around the globe.",
  <Japan>);
 manufacturer3: Manufacturer := new Manufacturer("Hyunday",
  "Hyundai Motor Company leads the Hyundai Motor Group, a wide-reaching business capable of
      processing resources from molten iron to finished cars.",
  <SouthKorea>);

 rally1: Rally := new Rally("The Dark Mountain", "A very dark mountain", <Germany>, <Mountain
     >, DateUtils`MakeDate(2017, 10, 5), DateUtils`MakeDate(2017, 10, 10));
 rally2: Rally := new Rally("Despair Woods", "Maze-like woods it the heart of Spain", <Spain>,
      <Forest>, DateUtils`MakeDate(2017, 10, 14), DateUtils`MakeDate(2017, 10, 20));
 rally3: Rally := new Rally("Fail Rally date", "Maze-like woods it the heart of Spain", <Spain
     >, <Forest>, DateUtils`MakeDate(2017, 9, 25), DateUtils`MakeDate(2017, 10, 20));

 car1: Car := new Car("Hyundai i20", manufacturer3, 370, 1250);
 car2: Car := new Car("Citroen C3", manufacturer1, 400, 1300);
 car3: Car := new Car("Toyota Yaris", manufacturer2, 480, 1200);
 car4: Car := new Car("Citroen C4", manufacturer1, 400, 1300);

 team1: Team := new Team("CITROEN Total Abu Dhabi WRT", <France>);
 team2: Team := new Team("Hyundai Motorsport", <Germany>);
 team3: Team := new Team("M-Sport World Rally Team", <GreatBritain>);
 team4: Team := new Team("TOYOTA Gazoo Racing WRT", <Japan>);

 driver1: Driver := new Driver("Dan", <Male>, <GreatBritain>, DateUtils`MakeDate(1980, 10, 5))
     ;
 driver2: Driver := new Driver("Joana", <Female>, <Spain>, DateUtils`MakeDate(1987, 2, 22));
 driver3: Driver := new Driver("Alexio", <Male>, <Germany>, DateUtils`MakeDate(1995, 5, 29));
 driver4: Driver := new Driver("Another one", <Male>, <Germany>, DateUtils`MakeDate(1996, 5,
     29));

 partner1: Partner := new Partner("Michelin", "Michelin, the leading tire company, is
     dedicated to enhancing its clients mobility, sustainably; designing and distributing the
     most suitable tires, services and solutions for its clients needs.");
 partner2: Partner := new Partner("Certina", "Precision, reliability, innovation and dynamism.
      These are the values that have always linked Certina to sport.");
 partner3: Partner := new Partner("OneBet", "OneBet is the official sports betting partner of
     the FIA World Rally Championship, providing spectacular opportunities to boost the rush
     and experience of WRC.");

 stage1: Stage := new Stage(1, <Special>, DateUtils`MakeDate(2017, 10, 6), 50);
 stage2: Stage := new Stage(2, <Special>, DateUtils`MakeDate(2017, 10, 8), 200);
 stage3: Stage := new Stage(1, <Special>, DateUtils`MakeDate(2017, 10, 15), 50);
 stage4: Stage := new Stage(2, <Special>, DateUtils`MakeDate(2017, 10, 16), 200);

 performance1: [Performance] := nil;
 performance2: [Performance] := nil;
 performance3: [Performance] := nil;
 performance4: [Performance] := nil;

operations

 public Run: () ==> ()
```

```
Run() == (
 IO`println("\nChampionship Tests");

 --Setup
 team1.AddCar(car1);
 team1.AddCar(car2);
 team2.AddCar(car3);
 team3.AddCar(car4);

 team1.AddDriver(driver1);
 team1.AddDriver(driver2);
 team2.AddDriver(driver3);
 team3.AddDriver(driver4);
 team1.AssignCarToDriver(car1, driver1);
 team1.AssignCarToDriver(car2, driver2);
 team2.AssignCarToDriver(car3, driver3);
 team3.AssignCarToDriver(car4, driver4);

 --Variables
 assertEqual("WRC", champ1.GetName());
 assertEqual(DateUtils`MakeDate(2017, 10, 2), champ1.GetStartDate());
 assertEqual(DateUtils`MakeDate(2017, 11, 20), champ1.GetEndDate());
 assertTrue(champ1.GetStartDate() < champ1.GetEndDate());

 --Partners
 champ1.AddPartner(partner1);
 champ1.AddPartner(partner2);
 assertTrue(partner2 in set champ1.GetPartners());
 champ1.RemovePartner(partner2);
 assertTrue(partner2 not in set champ1.GetPartners());

 --Teams
 champ1.AddTeam(team1);
 champ1.AddTeam(team2);
 assertTrue(team1 in set rng champ1.GetTeams());
 champ1.RemoveTeam(team1);
 assertEqual({ team2.GetName() |-> team2 }, champ1.GetTeams());
 champ1.AddTeam(team3);
 champ1.AddTeam(team1);
 --Team by name
 assertEqual(team2, champ1.GetTeamByName("Hyundai Motorsport"));

 --Rally
 rally1.AddStage(stage1);
 rally1.AddStage(stage2);
 rally2.AddStage(stage3);
 rally2.AddStage(stage4);
 champ1.AddRally(rally1);
 champ1.AddRally(rally2);
 assertEqual(rally2, champ1.GetRallyByName("Despair Woods"));
 assertTrue(let i in set inds champ1.GetSeries() in champ1.GetSeries()(i)=rally1);

 assertTrue({driver1, driver2, driver3, driver4} subset champ1.GetDrivers());

 assertEqual(nil, champ1.GetCurrentRally());
 champ1.StartChampionship();
 assertTrue(champ1.IsCurrentRallyCompleted() = false);
 assertEqual(rally1, champ1.GetCurrentRally());

 performance1 := new Performance(driver1);
 performance2 := new Performance(driver2);
 performance3 := new Performance(driver3);
 performance4 := new Performance(driver4);
 performance1.SetTime(200);
 performance1.SetAverageSpeed(50);
 performance2.SetTime(217);
 performance2.SetAverageSpeed(48);
 performance3.SetTime(202);
 performance3.SetAverageSpeed(50);
 performance4.SetTime(250);
 performance4.SetAverageSpeed(46);
 champ1.UpdateCurrentRally({performance1, performance2, performance3, performance4});
 champ1.UpdateCurrentRally({performance1, performance2, performance3, performance4});

 assertEqual(834, champ1.GetTeamRallyRanking("CITROEN Total Abu Dhabi WRT", "The Dark
     Mountain"));
 assertEqual(834, champ1.GetTeamGlobalRanking("CITROEN Total Abu Dhabi WRT"));

 assertTrue(champ1.IsCurrentRallyCompleted() = true);
 champ1.NextRally();
 champ1.UpdateCurrentRally({performance1, performance2, performance3, performance4});

 assertEqual(1251, champ1.GetTeamGlobalRanking("CITROEN Total Abu Dhabi WRT"));

 champ1.UpdateCurrentRally({performance1, performance2, performance3, performance4});
 assertTrue(champ1.IsCurrentRallyCompleted() = true);
 champ1.NextRally();
 assertEqual(<Completed>, champ1.GetState());

 assertTrue({"CITROEN Total Abu Dhabi WRT", "Hyundai Motorsport", "M-Sport World Rally Team"}
      subset dom champ1.GetTeamsRanking());
 assertTrue({1668, 808, 1000} subset rng champ1.GetTeamsRanking());
 IO`println("\nFinalizing Championship Tests");
);


 public testRalliesDates: () ==> ()
  -- A championship has a start and end date, just like a rally; A stage has a single date;
  -- So, a rally must have their dates between the respective championship that is part of;
      and a stage must have the date between the rally dates from which it belong to
  testRalliesDates() == (
   rally3.AddStage(stage1);
   rally3.AddStage(stage2);
   champ1.AddRally(rally3);
  );


 public testChampionshipStates: () ==> ()
  -- A championship also has a pre defined flow:
  -- As usual, all teams, partners and rallies are added; After this, they cannot be updated
  -- Then, the method StartChampionship is called, which initializes the first rally rankings
  -- The rally flow is then used, by calling the method UpdateCurrentRally
  -- When all the rally stages are completed and only then the NextRally must be called
  -- After all the rallies are completed, the endChampionship can be called.
  testChampionshipStates() == (
   team1.AddCar(car1);
   team1.AddCar(car2);
   team2.AddCar(car3);
   team3.AddCar(car4);
   team1.AddDriver(driver1);
   team1.AddDriver(driver2);
   team2.AddDriver(driver3);
   team3.AddDriver(driver4);
   team1.AssignCarToDriver(car1, driver1);
   team1.AssignCarToDriver(car2, driver2);
   team2.AssignCarToDriver(car3, driver3);
   team3.AssignCarToDriver(car4, driver4);
   --Teams
   champ1.AddTeam(team1);
   champ1.AddTeam(team2);
   champ1.AddTeam(team3);

   --Rally
   rally1.AddStage(stage1);
   rally1.AddStage(stage2);
   rally2.AddStage(stage3);
   rally2.AddStage(stage4);
   champ1.AddRally(rally1);
   champ1.AddRally(rally2);

   champ1.StartChampionship();
```

```
    performance1 := new Performance(driver1);
    performance2 := new Performance(driver2);
    performance3 := new Performance(driver3);
    performance4 := new Performance(driver4);
    performance1.SetTime(200);
    performance1.SetAverageSpeed(50);
    performance2.SetTime(217);
    performance2.SetAverageSpeed(48);
    performance3.SetTime(202);
    performance3.SetAverageSpeed(50);
    performance4.SetTime(250);
    performance4.SetAverageSpeed(46);

    champ1.UpdateCurrentRally({performance1, performance2, performance3, performance4});
    champ1.NextRally(); -- The current rally is not completed (there are stages to be ran)
    champ1.UpdateCurrentRally({performance1, performance2, performance3, performance4});
   );

end ChampionshipTest
```

Function or operation	Line	Coverage	Calls
AddPartner	102	100.0%	3
AddRally	180	100.0%	4
AddTeam	189	100.0%	6
Championship	31	100.0%	5
EndChampionship	91	100.0%	1
GetCurrentRally	213	100.0%	3
GetDriverGlobalRanking	125	100.0%	1
GetDrivers	226	100.0%	5
GetEndDate	255	100.0%	2
GetName	235	100.0%	1
GetPartners	275	100.0%	2
GetRallyByName	305	100.0%	8
GetSeries	285	100.0%	2
GetStartDate	245	100.0%	2
GetTeamByName	295	100.0%	2
GetTeamGlobalRanking	140	100.0%	29
GetTeamRallyRanking	114	100.0%	2
GetTeams	265	100.0%	2
GetTeamsRanking	157	100.0%	7
IsCurrentRallyCompleted	70	100.0%	12
NextRally	81	100.0%	3
RemovePartner	169	100.0%	1
RemoveTeam	199	100.0%	1
StartChampionship	46	100.0%	2
UpdateCurrentRally	59	100.0%	8
Championship.vdmpp		100.0%	114

4.3 DriverTest

```
class DriverTest is subclass of MyTestCase

instance variables
 team1: Team := new Team("The Awesome Ones", <GreatBritain>);
 team2: Team := new Team("The Better Ones", <Spain>);
 manufacturer1: Manufacturer := new Manufacturer("Hyunday", "Hyundai Motor Company", <
     SouthKorea>);
 manufacturer2: Manufacturer := new Manufacturer("Citroen", "Citroen", <France>);
 manufacturer3: Manufacturer := new Manufacturer("Toyota", "Toyota", <Japan>);

 car1: Car := new Car("Hyundai i20", manufacturer1, 370, 1250);
 driver1: Driver := new Driver("Dan", <Male>, <GreatBritain>, DateUtils`MakeDate(1980, 10, 5))
     ;

 car2: Car := new Car("Citroen C3", manufacturer2, 400, 1300);
 driver2: Driver := new Driver("Joana", <Female>, <Spain>, DateUtils`MakeDate(1987, 2, 22));

 car3: Car := new Car("Toyota Yaris", manufacturer3, 480, 1200);
 driver3: Driver := new Driver("Alexio", <Male>, <Germany>, DateUtils`MakeDate(1995, 5, 29));

operations

 public Run: () ==> ()
 Run() == (
  IO`println("\nDriver Tests");

  -- Setup
  team1.SetBacker(manufacturer1);
  team1.AddDriver(driver1);
  team1.AddCar(car1);
  team1.AssignCarToDriver(car1, driver1);

  team2.AddDriver(driver2);
  team2.AddCar(car2);
  team2.AssignCarToDriver(car2, driver2);

  team2.AddDriver(driver3);
  team2.AddCar(car3);
  team2.AssignCarToDriver(car3, driver3);

  -- Team
  assertEqual(team1, driver1.GetTeam());
  assertEqual(team2, driver2.GetTeam());

  -- Date
  assertEqual(DateUtils`MakeDate(1980, 10, 5), driver1.GetBirth());
  assertEqual(DateUtils`MakeDate(1995, 5, 29), driver3.GetBirth());

  -- Nationality
  assertEqual(<Spain>, driver2.GetNationality());
  assertEqual(<Germany>, driver3.GetNationality());

  -- Gender
  assertEqual(<Female>, driver2.GetGender());
  assertEqual(<Male>, driver3.GetGender());

  -- Name
  assertEqual("Dan", driver1.GetName());
  assertEqual("Joana", driver2.GetName());

  -- Car
  assertEqual(car1, driver1.GetCar());
  driver1.RemoveCar();
  assertEqual(nil, driver1.GetCar());
  assertEqual(car3, driver3.GetCar());

  -- Team
  assertEqual(team1, driver1.GetTeam());
  assertEqual(team2, driver2.GetTeam());

  IO`println("\nFinalizing Driver Tests");
 );
end DriverTest
```

Function or operation	Line	Coverage	Calls
Driver	27	100.0%	24
GetBirth	132	100.0%	2
GetCar	92	100.0%	232
GetGender	112	100.0%	2
GetName	102	100.0%	3204
GetNationality	122	100.0%	2
GetTeam	82	100.0%	300
RemoveCar	61	100.0%	3
RemoveTeam	53	100.0%	1
SetCar	69	100.0%	22
SetTeam	44	100.0%	23
Driver.vdmpp		100.0%	3815

4.4 ManufacturerTest

```
class ManufacturerTest is subclass of MyTestCase

instance variables
 manufacturer1: Manufacturer := new Manufacturer("Citroen",
  "Citroen is one of the world's leading mainstream car manufacturers.",
  <France>);
 manufacturer2: Manufacturer := new Manufacturer("Toyota",
  "Toyota is one of the worlds best-known and most successful businesses, building cars and
      trucks in 26 countries for sale in more than 160 markets around the globe.",
  <Japan>);
 manufacturer3: Manufacturer := new Manufacturer("Hyunday",
  "Hyundai Motor Company leads the Hyundai Motor Group, a wide-reaching business capable of
      processing resources from molten iron to finished cars.",
  <SouthKorea>);

operations

 public Run: () ==> ()
 Run() == (
  IO`println("\nManufacturer Tests");

  assertEqual("Citroen", manufacturer1.GetName());
  assertEqual("Toyota is one of the worlds best-known and most successful businesses, building
       cars and trucks in 26 countries for sale in more than 160 markets around the globe.",
      manufacturer2.GetDescription());
  assertEqual(<SouthKorea>, manufacturer3.GetCountry());

  IO`println("\nFinalizing Manufacturer Tests");
 );

end ManufacturerTest
```

Function or operation	Line	Coverage	Calls
GetCountry	51	100.0%	1
GetDescription	41	100.0%	1
GetName	31	100.0%	7
Manufacturer	16	100.0%	27
Manufacturer.vdmpp		100.0%	36

4.5 PerformanceTest

```
class PerformanceTest is subclass of MyTestCase

instance variables
 team0: Team := new Team("Test Team", <Germany>);
 driver1: Driver := new Driver("Anne", <Female>, <Germany>, DateUtils`MakeDate(1990, 8, 7));
 driver2: Driver := new Driver("Anna", <Female>, <Germany>, DateUtils`MakeDate(1990, 8, 7));
 driver3: Driver := new Driver("Annie", <Female>, <Germany>, DateUtils`MakeDate(1990, 8, 7));

 performance1: [Performance] := nil;
 performance2: [Performance] := nil;
 performance3: [Performance] := nil;

 ranking1: [Ranking] := nil;

 manufacturer1: Manufacturer := new Manufacturer("Citroen",
  "Citroen is one of the world's leading mainstream car manufacturers.",
  <France>);
 manufacturer2: Manufacturer := new Manufacturer("Toyota",
  "Toyota is one of the worlds best-known and most successful businesses, building cars and
      trucks in 26 countries for sale in more than 160 markets around the globe.",
  <Japan>);
 manufacturer3: Manufacturer := new Manufacturer("Hyunday",
  "Hyundai Motor Company leads the Hyundai Motor Group, a wide-reaching business capable of
      processing resources from molten iron to finished cars.",
  <SouthKorea>);

 car1: Car := new Car("Hyundai i20", manufacturer3, 370, 1250);
 car2: Car := new Car("Citroen C3", manufacturer1, 400, 1300);
 car3: Car := new Car("Toyota Yaris", manufacturer2, 480, 1200);

operations

 public Run: () ==> ()
  Run() == (
   IO`println("\nPerformance and Ranking Tests");

   --Setup
   team0.AddDriver(driver1);
   team0.AddDriver(driver2);
   team0.AddDriver(driver3);
   team0.AddCar(car1);
   team0.AddCar(car2);
   team0.AddCar(car3);
   team0.AssignCarToDriver(car1, driver1);
   team0.AssignCarToDriver(car2, driver2);
   team0.AssignCarToDriver(car3, driver3);

   -- Performance
   performance1 := new Performance(driver1);
   performance2 := new Performance(driver2);
   performance3 := new Performance(driver3);
   performance1.SetTime(200);
   performance1.SetAverageSpeed(50);
   performance2.SetTime(217);
   performance2.SetAverageSpeed(48);
   performance2.SetPenalty(1);
   performance3.SetTime(202);
   performance3.SetAverageSpeed(50);

   --Avg Speed
   assertEqual(50, performance1.GetAverageSpeed());
   assertEqual(48, performance2.GetAverageSpeed());

   --Penalty
   assertEqual(1, performance2.GetPenalty());
   assertEqual(0, performance3.GetPenalty());

   --Driver
   assertEqual(driver1, performance1.GetDriver());
   assertEqual(driver3, performance3.GetDriver());
   --Time
   assertEqual(200, performance1.GetTime());
   assertEqual(217, performance2.GetTime());

   --Comparison (TODO?)

   -- Ranking
   ranking1 := new Ranking(driver1);
   assertEqual(driver1, ranking1.GetDriver());
   assertEqual(0, ranking1.GetPoints());

   ranking1.AddPoints(50);
   assertEqual(50, ranking1.GetPoints());
   ranking1.AddPoints(-20);
   assertEqual(30, ranking1.GetPoints());

   IO`println("\nFinalizing Performance and Ranking Tests");
  );

end PerformanceTest
```

Function or operation	Line	Coverage	Calls
GetAverageSpeed	37	100.0%	2
GetDriver	27	100.0%	537
GetPenalty	73	100.0%	2
GetTime	55	100.0%	402
Performance	15	100.0%	18
SetAverageSpeed	45	100.0%	14
SetPenalty	81	100.0%	2
SetTime	63	0.0%	0
Performance.vdmpp		100.0%	977

Function or operation	Line	Coverage	Calls
AddPoints	23	100.0%	42
GetDriver	34	100.0%	3102
GetPoints	44	100.0%	242
Ranking	13	100.0%	21
Ranking.vdmpp		100.0%	3407

4.6 RallyTest

```
class RallyTest is subclass of MyTestCase

instance variables
 sponsor1: Sponsor := new Sponsor("Abu Dhabi", "Official visitor website for Abu Dhabi travel
     and tourism, offering information on hotels, restaurants, things to do, culture &
     heritage and events.");
 sponsor2: Sponsor := new Sponsor("Eparco", "Specialiste assainissement compact, developpe et
     fabrique en exclusivite des solutions pour lassainissement non collectif depuis 45 ans.")
     ;
 sponsor3: Sponsor := new Sponsor("Stilo", "Stilo helmets competition. Top functionality,
     everything a driver needs must be standard feature: intercom, connections, earmuffs,
     drinking system.");
 sponsor4: Sponsor := new Sponsor("Michelin", "Michelin, the leading tire company, is
     dedicated to enhancing its clients mobility, sustainably; designing and distributing the
     most suitable tires, services and solutions for its clients needs.");

 manufacturer1: Manufacturer := new Manufacturer("Citroen",
  "Citroen is one of the world’s leading mainstream car manufacturers.",
  <France>);
 manufacturer2: Manufacturer := new Manufacturer("Toyota",
  "Toyota is one of the worlds best-known and most successful businesses, building cars and
      trucks in 26 countries for sale in more than 160 markets around the globe.",
  <Japan>);
 manufacturer3: Manufacturer := new Manufacturer("Hyunday",
  "Hyundai Motor Company leads the Hyundai Motor Group, a wide-reaching business capable of
      processing resources from molten iron to finished cars.",
  <SouthKorea>);

 car1: Car := new Car("Hyundai i20", manufacturer3, 370, 1250);
 car2: Car := new Car("Citroen C3", manufacturer1, 400, 1300);
 car3: Car := new Car("Toyota Yaris", manufacturer2, 480, 1200);
 car4: Car := new Car("Citroen C4", manufacturer1, 400, 1300);

 team1: Team := new Team("CITROEN Total Abu Dhabi WRT", <France>);
 team2: Team := new Team("Hyundai Motorsport", <Germany>);
 team3: Team := new Team("M-Sport World Rally Team", <GreatBritain>);
 team4: Team := new Team("TOYOTA Gazoo Racing WRT", <Japan>);

 driver1: Driver := new Driver("Dan", <Male>, <GreatBritain>, DateUtils`MakeDate(1980, 10, 5))
     ;
 driver2: Driver := new Driver("Joana", <Female>, <Spain>, DateUtils`MakeDate(1987, 2, 22));
 driver3: Driver := new Driver("Alexio", <Male>, <Germany>, DateUtils`MakeDate(1995, 5, 29));
 driver4: Driver := new Driver("Another one", <Male>, <Germany>, DateUtils`MakeDate(1996, 5,
     29));

 stage1: Stage := new Stage(1, <Special>, DateUtils`MakeDate(2018, 1, 25), 50);
 stage2: Stage := new Stage(2, <Special>, DateUtils`MakeDate(2018, 1, 26), 200);

 rally1: Rally := new Rally("Rallye Monte-Carlo", "The Monte Carlo Rally or Rallye Monte Carlo
      is a rallying event organised each year by the Automobile Club de Monaco.", <France>, <
     Asphalt>, DateUtils`MakeDate(2018, 1, 25), DateUtils`MakeDate(2018, 1, 28));
 rally2: Rally := new Rally("Vodafone Rally de Portugal", "O Rali de Portugal e a maior prova
     de desporto motorizado que se realiza em Portugal anualmente.", <Portugal>, <Gravel>,
     DateUtils`MakeDate(2018, 5, 17), DateUtils`MakeDate(2018, 5, 20));
 rally3: Rally := new Rally("Rally Italia Sardegna", "Held on the beautiful island of Sardinia
     , Rally Italy - or Rally Italia Sardegna to give it its full name - is a notoriously
     tricky event.", <Italy>, <Mountain>, DateUtils`MakeDate(2018, 6, 7), DateUtils`MakeDate
     (2018, 6, 10));

 ranking1: [Ranking] := nil;
 ranking2: [Ranking] := nil;
 ranking3: [Ranking] := nil;
 ranking4: [Ranking] := nil;

 performance1: [Performance] := nil;
 performance2: [Performance] := nil;
 performance3: [Performance] := nil;
 performance4: [Performance] := nil;

operations

 public Run: () ==> ()
 Run() == (
  IO`println("\nRally Tests");

  --Setup
  team1.AddCar(car1);
  team1.AddCar(car2);
  team2.AddCar(car3);
  team3.AddCar(car4);

  team1.AddDriver(driver1);
  team1.AddDriver(driver2);
  team2.AddDriver(driver3);
  team3.AddDriver(driver4);
  team1.AssignCarToDriver(car1, driver1);
  team1.AssignCarToDriver(car2, driver2);
  team2.AssignCarToDriver(car3, driver3);
  team3.AssignCarToDriver(car4, driver4);
  --Variables
  assertEqual("Rallye Monte-Carlo", rally1.GetName());
  assertEqual("O Rali de Portugal e a maior prova de desporto motorizado que se realiza em
      Portugal anualmente.", rally2.GetDescription());
  assertEqual(<Italy>, rally3.GetCountry());
  assertEqual(<Gravel>, rally2.GetSurface());
  assertTrue(rally1.GetStartDate() < rally1.GetEndDate());
  assertEqual(0.0, rally2.GetDistance());

  --Stages
  rally1.AddStage(stage1);
  rally1.AddStage(stage2);
  assertEqual(250, rally1.GetDistance());
  assertEqual([stage1, stage2], rally1.GetStages());

  rally1.StartRally({driver1, driver2, driver3, driver4});
  assertTrue(len rally1.GetRankings() = 4);

  performance1 := new Performance(driver1);
  performance2 := new Performance(driver2);
  performance3 := new Performance(driver3);
  performance4 := new Performance(driver4);
  performance1.SetTime(200);
  performance1.SetAverageSpeed(50);
  performance2.SetTime(217);
  performance2.SetAverageSpeed(48);
  performance3.SetTime(202);
  performance3.SetAverageSpeed(50);
  performance4.SetTime(250);
  performance4.SetAverageSpeed(46);

  rally1.NextStage({performance1, performance2, performance3, performance4});

  assertEqual(stage2, rally1.GetCurrentStage());

  rally1.NextStage({performance1, performance2, performance3, performance4});

  assertTrue(rally1.GetDriverRanking(driver1).GetDriver() = driver1);
  assertTrue(rally1.GetDriverRanking(driver1).GetPoints() = 400 );

  assertTrue(rally1.GetDriverPerformanceOnStage(1, driver1).GetTime() = 200);

  assertEqual(834, rally1.GetTeamRanking("CITROEN Total Abu Dhabi WRT"));

  IO`println("\nFinalizing Rally Tests");
 );


 public testRallyStates: () ==> ()
  -- The occurrence of a rally and its stages works following this flow:
  -- A rally is started, and the participating drivers are defined;
  -- The next stage method is called, which finalizes a stage and defines its performances and
       initializes the next stage
  -- When it gets to the last stage, the rally is terminated and then no more changes can be
      done to the rankings or stage performances
  -- So, no stage can be terminated except from the NextStage method. There is a defined flow
      that must be followed
  testRallyStates() == (
   team1.AddCar(car1);
   team1.AddCar(car2);
   team2.AddCar(car3);
   team3.AddCar(car4);

   team1.AddDriver(driver1);
   team1.AddDriver(driver2);
   team2.AddDriver(driver3);
   team3.AddDriver(driver4);
   team1.AssignCarToDriver(car1, driver1);
   team1.AssignCarToDriver(car2, driver2);
   team2.AssignCarToDriver(car3, driver3);
   team3.AssignCarToDriver(car4, driver4);

   performance1 := new Performance(driver1);
   performance2 := new Performance(driver2);
   performance3 := new Performance(driver3);
   performance4 := new Performance(driver4);

   --Stages
   rally1.AddStage(stage1);
   rally1.AddStage(stage2);

   rally1.StartRally({driver1, driver2, driver3, driver4});

   stage1.Finalize();
   rally1.NextStage({performance1, performance2, performance3, performance4});
  );

end RallyTest
```

Function or operation	Line	Coverage	Calls
AddStage	122	100.0%	10
BubbleSort	261	100.0%	10
EndRally	94	100.0%	5
GetCountry	177	100.0%	1
GetCurrentStage	227	100.0%	5
GetDescription	167	100.0%	1
GetDistance	197	100.0%	2
GetDriverPerformanceOnStage	248	100.0%	3
GetDriverRanking	238	100.0%	4
GetEndDate	147	100.0%	267
GetName	157	100.0%	533
GetRankings	217	100.0%	1
GetStages	207	100.0%	5
GetStartDate	137	100.0%	527
GetSurface	187	100.0%	1
GetTeamRanking	107	100.0%	32
NextStage	73	100.0%	60
Rally	36	100.0%	8
StartRally	56	100.0%	5
Rally.vdmpp		100.0%	1480

4.7 SponsorTest

```
class SponsorTest is subclass of MyTestCase

instance variables
 sponsor1: Sponsor := new Sponsor("Abu Dhabi", "Official visitor website for Abu Dhabi travel
     and tourism, offering information on hotels, restaurants, things to do, culture &
     heritage and events.");
 sponsor2: Sponsor := new Sponsor("Eparco", "Specialiste assainissement compact, developpe et
     fabrique en exclusivite des solutions pour lassainissement non collectif depuis 45 ans.")
     ;
 sponsor3: Sponsor := new Sponsor("Stilo", "Stilo helmets competition. Top functionality,
     everything a driver needs must be standard feature: intercom, connections, earmuffs,
     drinking system.");
 sponsor4: Sponsor := new Sponsor("Michelin", "Michelin, the leading tire company, is
     dedicated to enhancing its clients mobility, sustainably; designing and distributing the
     most suitable tires, services and solutions for its clients needs.");
 partner1: Partner := new Partner("Michelin", "Michelin, the leading tire company, is
     dedicated to enhancing its clients mobility, sustainably; designing and distributing the
     most suitable tires, services and solutions for its clients needs.");
 partner2: Partner := new Partner("Certina", "Precision, reliability, innovation and dynamism.
      These are the values that have always linked Certina to sport.");
 partner3: Partner := new Partner("OneBet", "OneBet is the official sports betting partner of
     the FIA World Rally Championship, providing spectacular opportunities to boost the rush
     and experience of WRC.");

operations

 public Run: () ==> ()
 Run() == (
  IO`println("\nSponsor and Partner Tests");

  assertEqual("Abu Dhabi", sponsor1.GetName());
  assertEqual("Specialiste assainissement compact, developpe et fabrique en exclusivite des
      solutions pour lassainissement non collectif depuis 45 ans.", sponsor2.GetDescription())
      ;

  assertEqual("Michelin", partner1.GetName());
  assertEqual("Precision, reliability, innovation and dynamism. These are the values that have
       always linked Certina to sport.", partner2.GetDescription());

  IO`println("\nFinalizing Sponsor and Partner Tests");
 );

end SponsorTest
```

Function or operation	Line	Coverage	Calls
GetDescription	38	100.0%	1
GetName	28	100.0%	7
Sponsor	14	100.0%	17
Sponsor.vdmpp		100.0%	25

Function or operation	Line	Coverage	Calls
GetDescription	38	100.0%	1
GetName	28	100.0%	7
Partner	14	100.0%	7
Partner.vdmpp		100.0%	15

4.8 StageTest

```
class StageTest is subclass of MyTestCase

instance variables
 stage1: Stage := new Stage(1, <Special>, DateUtils`MakeDate(2017, 12, 1), 10);
 stage2: Stage := new Stage(2, <Special>, DateUtils`MakeDate(2017, 12, 2), 50);
 stage3: Stage := new Stage(3, <Special>, DateUtils`MakeDate(2017, 12, 3), 3);
 stage4: Stage := new Stage(4, <Transport>, DateUtils`MakeDate(2017, 12, 5), 100);

 team0: Team := new Team("Test Team", <Germany>);
 driver1: Driver := new Driver("Anne", <Female>, <Germany>, DateUtils`MakeDate(1990, 8, 7));
 driver2: Driver := new Driver("Anna", <Female>, <Germany>, DateUtils`MakeDate(1990, 8, 7));
 driver3: Driver := new Driver("Annie", <Female>, <Germany>, DateUtils`MakeDate(1990, 8, 7));

 performance1: [Performance] := nil;
 performance2: [Performance] := nil;
 performance3: [Performance] := nil;

 manufacturer1: Manufacturer := new Manufacturer("Citroen",
  "Citroen is one of the world's leading mainstream car manufacturers.",
  <France>);
 manufacturer2: Manufacturer := new Manufacturer("Toyota",
  "Toyota is one of the worlds best-known and most successful businesses, building cars and
      trucks in 26 countries for sale in more than 160 markets around the globe.",
  <Japan>);
 manufacturer3: Manufacturer := new Manufacturer("Hyunday",
  "Hyundai Motor Company leads the Hyundai Motor Group, a wide-reaching business capable of
      processing resources from molten iron to finished cars.",
  <SouthKorea>);

 car1: Car := new Car("Hyundai i20", manufacturer3, 370, 1250);
 car2: Car := new Car("Citroen C3", manufacturer1, 400, 1300);
 car3: Car := new Car("Toyota Yaris", manufacturer2, 480, 1200);

operations

 public Run: () ==> ()
 Run() == (
  IO`println("\nStage Tests");

  --Setup
  team0.AddDriver(driver1);
  team0.AddDriver(driver2);
  team0.AddDriver(driver3);
  team0.AddCar(car1);
  team0.AddCar(car2);
  team0.AddCar(car3);
  team0.AssignCarToDriver(car1, driver1);
  team0.AssignCarToDriver(car2, driver2);
  team0.AssignCarToDriver(car3, driver3);
  performance1 := new Performance(driver1);
  performance2 := new Performance(driver2);
  performance3 := new Performance(driver3);

  performance1.SetTime(200);
  performance1.SetAverageSpeed(50);

  performance2.SetTime(217);
  performance2.SetAverageSpeed(48);
  performance2.SetPenalty(1);

  performance3.SetTime(202);
  performance3.SetAverageSpeed(52);

  --Id
  assertEqual(1, stage1.GetId());
  assertEqual(2, stage2.GetId());

  --State
  assertEqual(<OffSeason>, stage1.GetState());
  stage1.Init();
  assertEqual(<Occurring>, stage1.GetState());
  stage1.Finalize();
  assertEqual(<Completed>, stage1.GetState());

  --Type
  assertEqual(<Special>, stage1.GetType());
  assertEqual(<Transport>, stage4.GetType());

  stage1.AddPerformance(performance1);
  stage1.AddPerformance(performance2);
  stage1.AddPerformance(performance3);

  --Date
  assertEqual(DateUtils`MakeDate(2017, 12, 3), stage3.GetDate());
  assertEqual(DateUtils`MakeDate(2017, 12, 5), stage4.GetDate());

  --Distance
  assertEqual(10, stage1.GetDistance());
  assertEqual(100, stage4.GetDistance());
  --Driver Performance
  assertEqual(performance1, stage1.GetDriverPerformance(driver1));
  assertEqual(nil, stage2.GetDriverPerformance(driver1));
    assertEqual(performance3, stage1.GetDriverPerformance(driver3));
    assertEqual(performance2, stage1.GetDriverPerformance(driver2.GetName()));
    assertEqual(nil, stage2.GetDriverPerformance(driver1.GetName()));

  --Performances
  assertEqual([performance1, performance3, performance2], stage1.GetPerformances());

  IO`println("\nFinalizing Stage Tests");
 );


 public testInvalidPerformance: () ==> ()
  -- Only if the stage is completed, is then possible to add a driver performance
  testInvalidPerformance() == (
   team0.AddDriver(driver1);
   team0.AddCar(car1);
   team0.AssignCarToDriver(car1, driver1);
   performance1 := new Performance(driver1);
   stage1.AddPerformance(performance1);
  );

end StageTest
```

Function or operation	Line	Coverage	Calls
AddPerformance	42	100.0%	43
Compare	133	100.0%	189
GetDate	101	100.0%	22
GetDistance	111	100.0%	12
GetDriverPerformance	53	100.0%	3
GetId	91	100.0%	828
GetPerformances	121	100.0%	9
GetType	81	100.0%	2
InsertSorted	127	100.0%	91
Stage	25	100.0%	14
Stage.vdmpp		100.0%	1213

4.9 TeamTest

```
class TeamTest is subclass of MyTestCase

instance variables
 sponsor1: Sponsor := new Sponsor("Abu Dhabi", "Official visitor website for Abu Dhabi travel
     and tourism, offering information on hotels, restaurants, things to do, culture &
     heritage and events.");
 sponsor2: Sponsor := new Sponsor("Eparco", "Specialiste assainissement compact, developpe et
     fabrique en exclusivite des solutions pour lassainissement non collectif depuis 45 ans.")
     ;
 sponsor3: Sponsor := new Sponsor("Stilo", "Stilo helmets competition. Top functionality,
     everything a driver needs must be standard feature: intercom, connections, earmuffs,
     drinking system.");
 sponsor4: Sponsor := new Sponsor("Michelin", "Michelin, the leading tire company, is
     dedicated to enhancing its clients mobility, sustainably; designing and distributing the
     most suitable tires, services and solutions for its clients needs.");

 manufacturer1: Manufacturer := new Manufacturer("Citroen",
  "Citroen is one of the world's leading mainstream car manufacturers.",
  <France>);
 manufacturer2: Manufacturer := new Manufacturer("Toyota",
  "Toyota is one of the worlds best-known and most successful businesses, building cars and
      trucks in 26 countries for sale in more than 160 markets around the globe.",
  <Japan>);
 manufacturer3: Manufacturer := new Manufacturer("Hyunday",
  "Hyundai Motor Company leads the Hyundai Motor Group, a wide-reaching business capable of
      processing resources from molten iron to finished cars.",
  <SouthKorea>);

 car1: Car := new Car("Hyundai i20", manufacturer3, 370, 1250);
 car2: Car := new Car("Citroen C3", manufacturer1, 400, 1300);
 car3: Car := new Car("Toyota Yaris", manufacturer2, 480, 1200);
 car4: Car := new Car("Citroen C4", manufacturer1, 400, 1300);

 team1: Team := new Team("CITROEN Total Abu Dhabi WRT", <France>);
 team2: Team := new Team("Hyundai Motorsport", <Germany>);
 team3: Team := new Team("M-Sport World Rally Team", <GreatBritain>);
 team4: Team := new Team("TOYOTA Gazoo Racing WRT", <Japan>);

 driver1: Driver := new Driver("Dan", <Male>, <GreatBritain>, DateUtils`MakeDate(1980, 10, 5))
     ;
 driver2: Driver := new Driver("Joana", <Female>, <Spain>, DateUtils`MakeDate(1987, 2, 22));
 driver3: Driver := new Driver("Alexio", <Male>, <Germany>, DateUtils`MakeDate(1995, 5, 29));

operations

 public Run: () ==> ()
 Run() == (
  IO`println("\nTeam Tests");

  assertEqual("CITROEN Total Abu Dhabi WRT", team1.GetName());
  assertEqual(<France>, team1.GetBased());
  assertEqual(nil, team1.GetBacker());
  assertEqual({}, team1.GetSponsors());

  team1.AddSponsor(sponsor1);
  assertTrue(sponsor1 in set team1.GetSponsors());
  team1.AddSponsor(sponsor2);
  assertTrue(sponsor2 in set team1.GetSponsors());
  team1.RemoveSponsor(sponsor1);
  assertTrue(sponsor1 not in set team1.GetSponsors());

  team1.AddCar(car1);
  team1.AddCar(car2);

  assertTrue({car1, car2} = team1.GetCars());
  team1.RemoveCar(car1);
  assertTrue(car1 not in set team1.GetCars());

  assertTrue(nil = team1.GetBacker());
  team1.SetBacker(manufacturer1);
  assertEqual(manufacturer1, team1.GetBacker());

  team1.AddCar(car4);
  team1.RemoveBacker();
  team1.AddCar(car3);

  team1.AddDriver(driver1);
  team1.AddDriver(driver2);
  assertTrue({driver1, driver2} = team1.GetDrivers());

  team1.RemoveDriver(driver1);
  assertTrue(driver1 not in set team1.GetDrivers());

  team1.AssignCarToDriver(car2, driver2);
  team1.RemoveCar(car3);

  team1.UnassignCarFromDriver(driver2);

  assertEqual(driver2, team1.GetDriverByName("Joana"));

  IO`println("\nFinalizing Team Tests");
 );


 public testInvalidTeamDriver: () ==> ()
  -- All the drivers that belong to a team, must have their team variable pointing to the
      respective team
  testInvalidTeamDriver() == (
   team1.AddDriver(driver1);
   driver1.RemoveTeam();
  );


  public testInvalidTeamDriverCar: () ==> ()
  -- All the drivers only can have a car if it's assigned by their own team. If a car is
      assigned and don't belong to the team, the invariant is violated
  testInvalidTeamDriverCar() == (
   team1.AddDriver(driver1);
   driver1.SetCar(car1);
  );


  public testInvalidTeamCar: () ==> ()
  -- If the team backer is instantiated, then all the cars that belong to the team need to
      have its manufacturer object pointing to the team backer.
  -- Otherwise, the team cars can have different manufacturers
  testInvalidTeamCar() == (
   team1.AddCar(car2);
   team1.AddCar(car4);
   team1.SetBacker(manufacturer1);
   team1.AddCar(car3);
  );

end TeamTest
```

Function or operation	Line	Coverage	Calls
Run	34	100.0%	1
testInvalidTeamCar	96	0.0%	0
testInvalidTeamDriver	82	0.0%	0
testInvalidTeamDriverCar	89	0.0%	0
TeamTest.vdmpp		91.8%	1

4.10 UseCasesTest

```
class UseCasesTest is subclass of MyTestCase

instance variables
 champ1: Championship := new Championship("European Championship", DateUtils`MakeDate(2017,
     10, 29), DateUtils`MakeDate(2017, 11, 16));

 rally1: Rally := new Rally("Rally da Costa", "Rally in Portugal", <Portugal>, <Sand>,
     DateUtils`MakeDate(2017, 11, 1), DateUtils`MakeDate(2017, 11, 3));
 rally2: Rally := new Rally("Rally de la Coruna", "Rally in Spain", <Spain>, <Asphalt>,
     DateUtils`MakeDate(2017, 11, 5), DateUtils`MakeDate(2017, 11, 15));

 stage1r1: Stage := new Stage(201711, <Transport>, DateUtils`MakeDate(2017, 11, 1), 350);
 stage2r1: Stage := new Stage(201712, <Special>, DateUtils`MakeDate(2017, 11, 3), 50);
 stage1r2: Stage := new Stage(201721, <Special>, DateUtils`MakeDate(2017, 11, 5), 10);
 stage2r2: Stage := new Stage(201722, <Transport>, DateUtils`MakeDate(2017, 11, 8), 300);

 manufacturer1: Manufacturer := new Manufacturer("Citroen", "Citroen", <France>);
 manufacturer2: Manufacturer := new Manufacturer("Toyota", "Toyota", <Japan>);
 manufacturer3: Manufacturer := new Manufacturer("Mitsubishi", "Mitsubishi", <Japan>);

 team1: Team := new Team("Team1", <Germany>);
 team2: Team := new Team("Team2", <SouthKorea>);

 car1t1: Car := new Car("Toyota Yaris", manufacturer2, 480, 1200);
 car2t1: Car := new Car("Citroen C3", manufacturer1, 400, 1300);
 car1t2: Car := new Car("Citroen C4", manufacturer1, 400, 1300);
 car2t2: Car := new Car("Mitsubishi Mirage R5", manufacturer3, 450, 1310);

 driver1t1: Driver := new Driver("Dan", <Male>, <GreatBritain>, DateUtils`MakeDate(1980, 10,
     5));
 driver2t1: Driver := new Driver("Joana", <Female>, <Spain>, DateUtils`MakeDate(1987, 2, 22));
 driver1t2: Driver := new Driver("Anne", <Female>, <Germany>, DateUtils`MakeDate(1990, 8, 7));
 driver2t2: Driver := new Driver("Anna", <Female>, <Germany>, DateUtils`MakeDate(1990, 8, 7));

 driver1t1perf: Performance;
 driver2t1perf: Performance;
 driver1t2perf: Performance;
 driver2t2perf: Performance;

operations

 public Run: () ==> ()
  Run() == (
   IO`println("\nUse Cases Tests");
   Setup();

   --Use Case: See car's information
   assertEqual(car1t1, driver1t1.GetCar());

   --Start Championship
   assertEqual(nil, champ1.GetCurrentRally());
   champ1.StartChampionship();
   assertEqual({"Team1" |-> 0, "Team2" |-> 0}, champ1.GetTeamsRanking()); --Use Case: See team
       's global performance

   --Rally 1
   assertEqual(rally1, champ1.GetCurrentRally());
   assertFalse(champ1.IsCurrentRallyCompleted());

   assertEqual(stage1r1, rally1.GetCurrentStage());
   champ1.UpdateCurrentRally({driver1t1perf, driver2t1perf, driver1t2perf, driver2t2perf}); --
       Use Case: Record relevant stage info
   assertEqual({"Team1" |-> 225, "Team2" |-> 230}, champ1.GetTeamsRanking()); --Use Case: See
       team's global performance

   assertEqual(stage2r1, rally1.GetCurrentStage());
   champ1.UpdateCurrentRally({driver1t1perf, driver2t1perf, driver1t2perf, driver2t2perf}); --
       Use Case: Record relevant stage info
   assertEqual({"Team1" |-> 450, "Team2" |-> 460}, champ1.GetTeamsRanking()); --Use Case: See
       team's global performance
   assertEqual(460, champ1.GetTeamRallyRanking("Team2", "Rally da Costa")); --Use Case: See
       team's rally ranking
   assertEqual(200, rally1.GetDriverRanking(driver1t1).GetPoints()); --Use Case: See driver's
       rally performance
   assertEqual(100, rally1.GetDriverPerformanceOnStage(201712, driver1t1).GetTime()); --Use
       Case: See driver's stage performance
   assertEqual(200, champ1.GetDriverGlobalRanking(driver1t1));--Use Case: See driver's global
       performance

   assertTrue(champ1.IsCurrentRallyCompleted());
   champ1.NextRally();

   --Rally 2
   assertEqual(rally2, champ1.GetCurrentRally());

   assertEqual(stage1r2, rally2.GetCurrentStage());
   champ1.UpdateCurrentRally({driver1t1perf, driver2t1perf, driver1t2perf, driver2t2perf}); --
       Use Case: Record relevant stage info
   assertEqual({"Team1" |-> 675, "Team2" |-> 690}, champ1.GetTeamsRanking()); --Use Case: See
       team's global performance

   assertEqual(stage2r2, rally2.GetCurrentStage());
```

```
    champ1.UpdateCurrentRally({driver1t1perf, driver2t1perf, driver1t2perf, driver2t2perf}); --
        Use Case: Record relevant stage info
    assertEqual({"Team1" |-> 900, "Team2" |-> 920}, champ1.GetTeamsRanking()); --Use Case: See
        team's global performance

    assertTrue(champ1.IsCurrentRallyCompleted());

    IO`println("\nFinalizing Use Cases Tests");
   );


 private Setup: () ==> ()
  Setup() == (
  --Setup championship's teams (Use Case: Manage teams)
  champ1.AddTeam(team1);
  champ1.AddTeam(team2);

  --Setup championship's partners (Use Case: Set partners)
  champ1.AddPartner(new Partner("Partner 1", "The only partner this championship needs!"));

  --Setup series' stages (Use Case: Set rallies and their stages)
  rally1.AddStage(stage1r1);
  rally1.AddStage(stage2r1);
  rally2.AddStage(stage1r2);
  rally2.AddStage(stage2r2);

  --Setup championship's rallies (Use Case: Set rallies and their stages)
  champ1.AddRally(rally1);
  champ1.AddRally(rally2);

  --Setup teams' (Use Case: Set sponsors)
  team1.AddSponsor(new Sponsor("Sp", "Team 1's only sponsor"));

  --Setup teams' cars (Use Case: Set drivers and their cars)
  team1.AddCar(car1t1);
  team1.AddCar(car2t1);
  team2.AddCar(car1t2);
  team2.AddCar(car2t2);

  --Setup teams' drivers (Use Case: Set drivers and their cars)
  team1.AddDriver(driver1t1);
  team1.AddDriver(driver2t1);
  team2.AddDriver(driver1t2);
  team2.AddDriver(driver2t2);

  --Setup driver's cars (Use Case: Set drivers and their cars)
  team1.AssignCarToDriver(car1t1, driver1t1);
  team1.AssignCarToDriver(car2t1, driver2t1);
  team2.AssignCarToDriver(car1t2, driver1t2);
  team2.AssignCarToDriver(car2t2, driver2t2);

  --Setup performances
  driver1t1perf := new Performance(driver1t1);
  driver2t1perf := new Performance(driver2t1);
  driver1t2perf := new Performance(driver1t2);
  driver2t2perf := new Performance(driver2t2);
  driver1t1perf.SetTime(100);
  driver2t1perf.SetTime(125);
  driver1t2perf.SetTime(110);
  driver2t2perf.SetTime(120);
  );

end UseCasesTest
```

5 Model verification

5.1 Domain verification

No.	PO Name	Type
67	Championship‘AddTeam(team)	legal map application

Table 15: Domain Verification

The code under analysis is:

```
public AddTeam: Team ==> ()
 AddTeam(t) == (
   teams := teams munion {t.GetName() |-> t};
 )
 pre state = <OffSeason> and t.GetName() not in set dom teams
 post teams = teams~ munion {t.GetName() |-> t};
```

In this case the proof is easy because the verification ***t.GetName() not in set dom teams*** ensures that the team was not already in ***teams***, not allowing the attempt to insert a duplicated team.

5.2 Invariant verification

No.	PO Name	Type
178	Team‘AddCar(Car)	state invariant holds

Table 16: Invariant Verification

The code under analysis is:

```
public AddCar: Car ==> ()
 AddCar(c) == (
   cars := cars union {c}
 )
 pre if(backer <> nil) then c.GetManufacturer().GetName() =
   backer.GetName() else true and not exists c1 in set cars
  & c1.GetModel() = c.GetModel()
 post c in set cars;
```

The relevant invariant under analysis is:

```
 inv if(backer <> nil) then forall c in set cars
  & c.GetManufacturer() = backer else true;
```

The pre condition, assures that if the team has a backer (manufacturer), then the car to be added must have the backer as manufacturer. By enforcing this condition,

6 Code generation

After the Java code generation, the group found three small problems, that were quickly fixed.

The first was due to an invalid type checking, where k.size() returned an int but the code generated expected a Long.

```
-- Operation BubbleSort, Rally.vdmpp
-- Original
      for (Long i  = k.size();
-- Modified
      for (Long i  = Long.valueOf(k.size());
```

The second problem was just an invalid static function call.

```
-- Operation AddPerformance, Stage.vdmpp
-- Original
      performances = InsertSorted(perf, Utils.copy(performances),
      Stage.Compare);
-- Modified
      performances = InsertSorted(perf, Utils.copy(performances),
      Stage::Compare);
```

The last problem was caused by the classes who had bidirectional associations. As such, the function ToString() printed the own class and their associations, but since the associations also referred the first class, there was a cyclic loop. It was necessary to remove those print calls.

Afterwards, we ran the tests created and the results were valid, just like in the Overture runs. Nevertheless, the group decided to create a GUI to thoroughly test the project main functions, where we had successful results. Mainly, the constructors, getters, removes, adds and setters functions were tested, and the system worked as expected, with valid inputs, of course, since the invariants, pre and post conditions weren't generated.

7 Conclusions

The project was concluded with all the predicted requirements achieved. A coverage of 100% was achieved, which gives us full confidence in our project. We also provide a GUI for better understanding of the core features of the project.

Further improvements could be made in terms of search functions and championship/rallies statistics, as they are important features for the real use of this tool.

Contribution:

- Diogo Duque - 50%
- Renato Abreu - 50%

References

[1] First specification, `https://play.google.com/store/apps/details?id=com.crml.android.rally`

[2] VDM slides, `https://moodle.up.pt/pluginfile.php/165033/mod_resource/content/0/VDM%2B%2B.pdf`

[3] Overture Quick Start, `https://moodle.up.pt/pluginfile.php/164632/mod_resource/content/0/OverturQuickStartExercise.pdf`

[4] VMD++ Reference, `https://raw.githubusercontent.com/overturetool/documentation/master/documentation/VDM10LangMan/VDM10_lang_man.pdf`