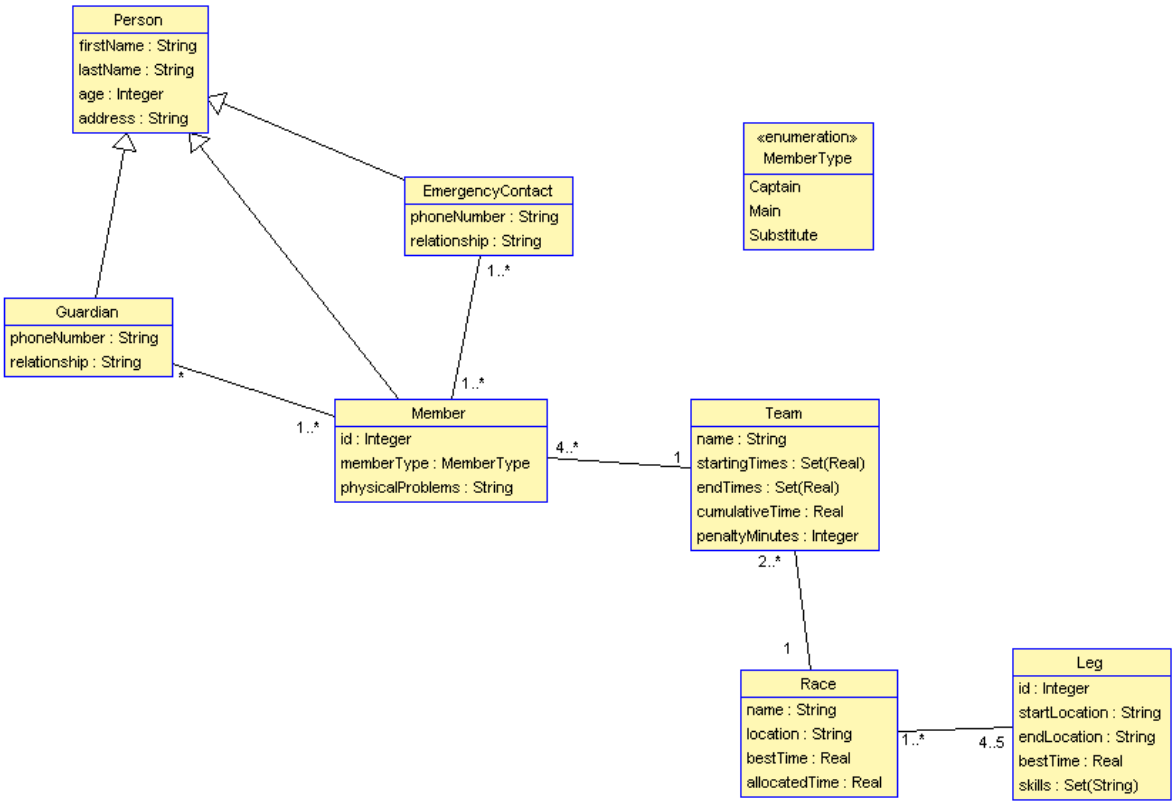
Assignment 3

1) Class Diagram

Here is the UML Class Diagram created using the USE tool:



Assumptions:

- A guardian must have at least 1 member participating in the race, but a member can have 0 guardians (if age > 18).
- An emergency contact must have at least 1 member participating in the race, and a member needs at least 1 emergency contact.
- A member can only be a part of 1 team.
- There must be at least 2 teams in a race.
- A leg must be a part of at least 1 race.

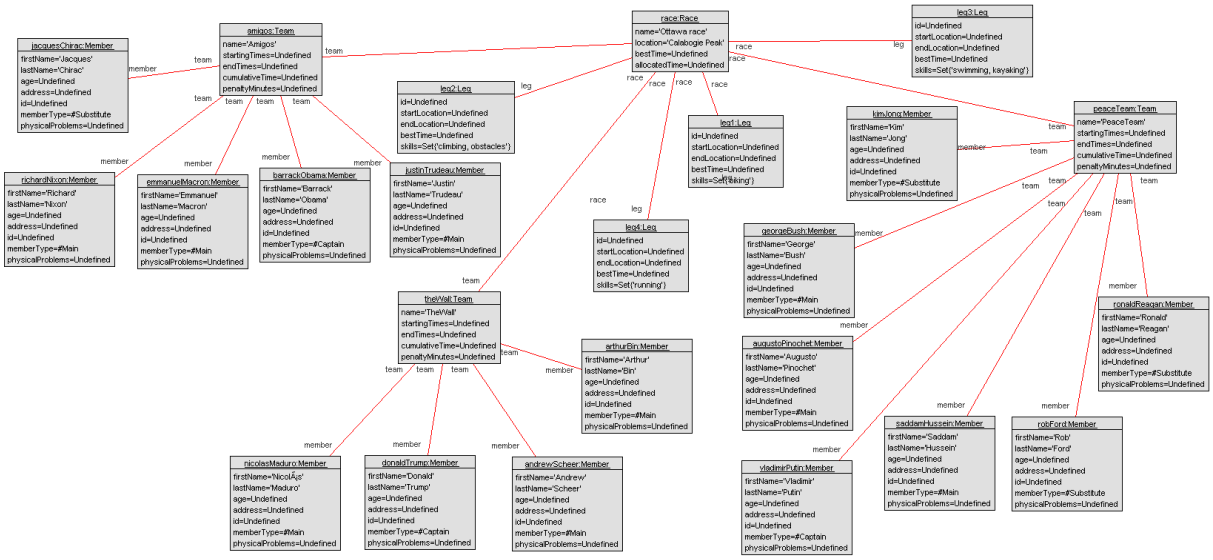
2) List of Operations

Based on the description of the system, here are 5 operations esteemed important for basic functionality:

- *substituteMember(Member main, Member sub)*: This operation, in the *Team* class, takes a main member and swaps them with a substitute member. This operation can only be performed by a team captain. Returns true if the swap was successful and false otherwise.
- *calculateScore()*: This operation, in the *Team* class, calculates the final score of a team after having completed the 4 or 5 legs of a race. It deducts the penalty minutes from the cumulative time and returns a final score in time format.
- *isValidMember()*: This operation, in the *Member* class, will be used to perform checks to validate a member joining a team. It will check that the member successfully submitted a signed form, an emergency contact info was provided and if the member is under 18, it will ensure a guardian or parent information was also provided. It will also ensure that the member doesn't already belong to another team. Returns true, if the member can join the team and false otherwise.
- *modifyTeamName(String newName)*: This operation, in the *Team* class, takes a string parameter which will be the new team name. This operation must check that the member performing the operation is the captain of the team. Returns true if the name change was successful and false otherwise.
- *addMember(Member member)*: This operation, in the *Team* class, takes a member and validates his entry in the team by the team captain. Returns true if successful and false otherwise.

3) Object Diagram

The object diagram was created according to the given specifications and was generated using the Objects.cmd file. The diagram is displayed below. It is not very readable due to the size of the object diagram.



4) OCL Constraints

The following OCL constraints were used:

- **Member first name cannot be empty:** context Member inv FirstNameNotEmpty: self.firstName.size() > 0
- **Member last name cannot be empty:** context Member inv LastNameNotEmpty: self.lastName.size() > 0
- **A race must include at least 4 legs:** context Race inv MinFourLegs: self.leg -> size >=4
- **A leg must have different targeted skills. That is, leg 1 and 2 cannot both target 'running':** context Leg inv DifferentSkills: Leg.allInstances -> forAll(l1, l2 | l1<>l2 implies l1.skills->intersection(l2.skills)->size = 0)
- **The score of the team (which correspond to the cumulative time of all members) cannot exceed the total time allocated to the race:** context Team inv TotalScore: self.cumulativeTime <= self.race.allocatedTime
- **A member cannot be part of more than one team:** context Member inv OneTeam: self.team -> size = 1
- **Member id must be unique: context Member inv UniqueID:** Member.allInstances() -> forAll(m1, m2 | m1 <> m2 implies m1.id <> m2.id)

- **A team name must be unique: context Team inv UniqueName:** Team.allInstances() -> forAll(t1, t2 | t1 <> t2 implies t1.name <> t2.name)
- **The starting time given to a team cannot be the same as the one given to another team (all team start with a 10-minute gap):** context Team inv UniqueLegTimes: Team.allInstances() -> forAll(t1, t2 | t1 <> t2 implies t1.startingTimes ->intersection(t2.startingTimes)->size = 0)