# November 2015 Lab – Football League

Design and implement **a football league holding teams, matches, scores and players.** Requirements:

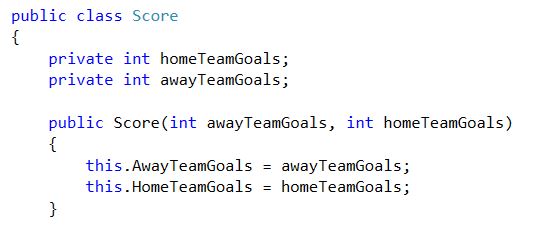
* Teams have Name (**string**), Nickname (**string**), Date of Founding (**datetime**) and a collection of Players
  + Name and Nickname should be at least 5 characters long
  + The collection of players cannot hold two identical players (where firstname and lastname match)
* Matches have Home Team, Away Team, Score, and Id (**int**)
  + The two teams should be different (identified by name)
* Scores have Away Team Goals (**int**) and Home Team Goals (**int**)
  + Goals cannot be negative
* Players have Firstname (**string**), Lastname (**string**), Salary (**decimal**), Date of Birth (**datetime**) and Team
  + Firstname and lastname must be at least 3 characters long
  + Salary cannot be negative
  + Date of Birth’s year cannot be lower than 1980
* There is only one league that holds a collection of matches and a collection of teams
  + Matches should be unique (identified by id)
  + Teams should be unique (identified by name)

### Step 1: Create the classes

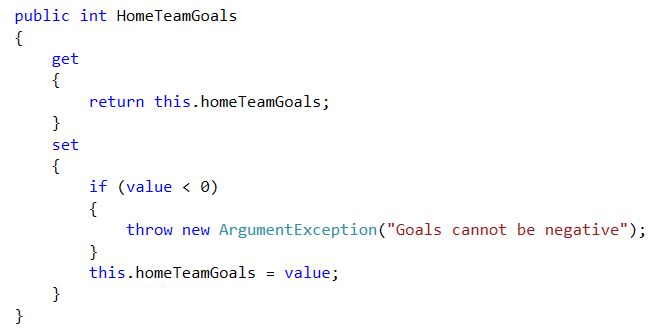
Create a console application called Football League. In it, add a folder called Models that will hold your classes. Add the described classes.

1. Make the League class static, we will have one league only.
2. ­­­The other classes should allow being instantiated more than once.
3. Add fields, constructors and properties.
4. Throw exceptions when the data is not correct
5. Override the ToString() method.

The Score class should be something like this

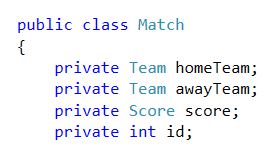


We need to add properties that validate whether the goals we are trying to set are correct. As you remember, we cannot negative goals



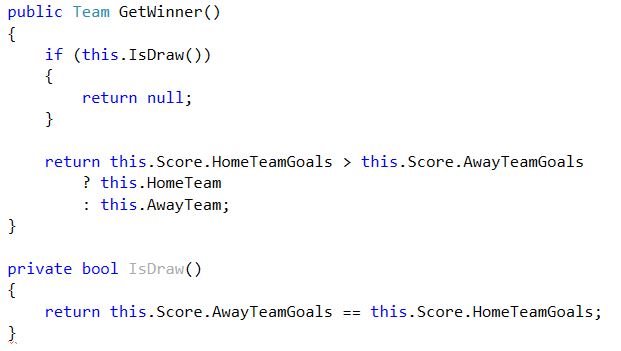
**Do exactly the same for AwayTeamGoals.**

Let’s continue with the other classes. The Match should hold two teams, a score, and a unique Id to identify it from other matches.

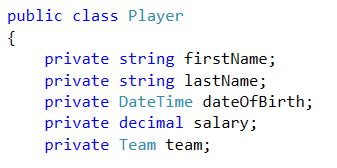


TODO: Create **properties** for the **fields** and initialize them from a **constructor**.

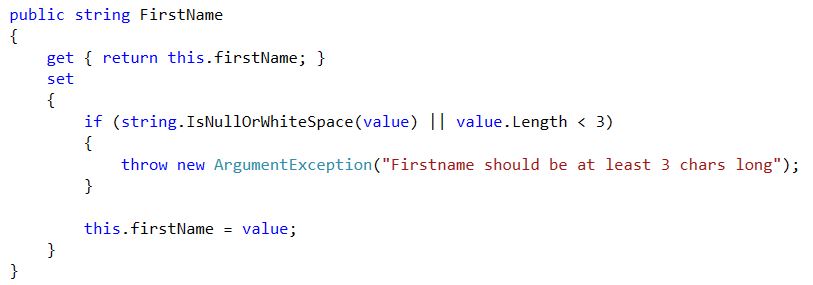
The match should also know its winner



The player class contains first name, last name, date of birth, salary and team.

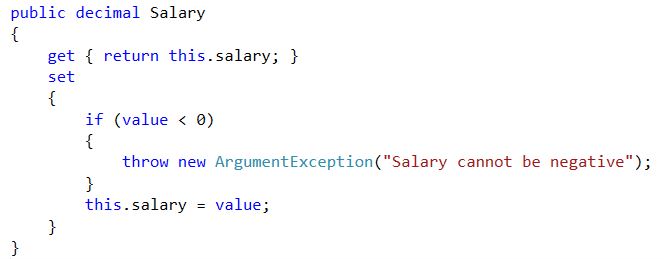


Initialize all the fields from a constructor. You need properties for this. Add validations in the properties.

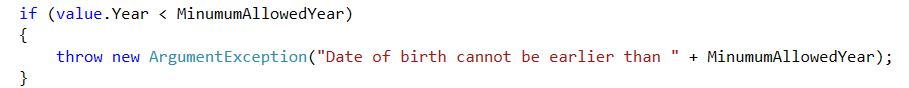


Do the same for the LastName property. Since the two validations are the same, you may want to create a method to do the checks.

The salary cannot be negative:



Add a property for the date of birth. It should not allow dates lower than 1980.



You will need a constant before this.

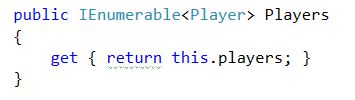
C:\Users\Edu\Pictures\const.JPG

You also need a property for the player’s Team.



Add properties. The Name and Nickname should have at least 5 characters. The DateFounded’s year should be after 1850.

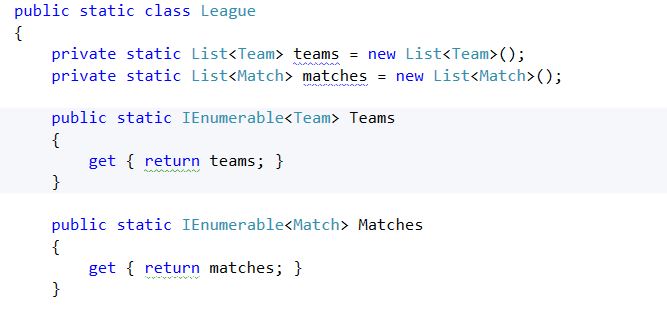
The interesting thing in the team class is the collection of players. We don’t want to be able to set it from outside the class, so it will not have a setter.



We will add players to the team through a special method.

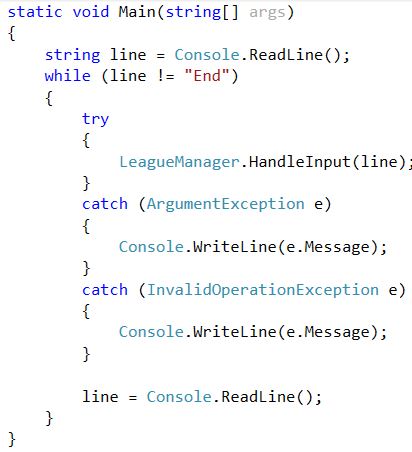


The league will hold a collection of teams and a collection of matches. We have only one league so we will make it static.

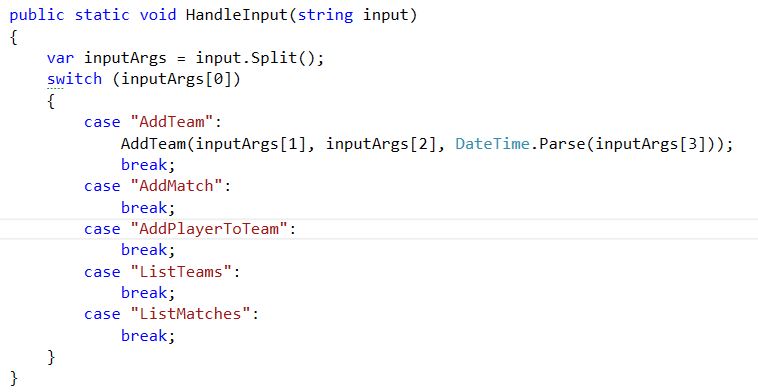


Create methods that add teams and matches. There cannot be two teams with the same **Name** in the collection. There cannot be two matches with the same **Id** in the other collection.

Now, let’s create a main method where we read the input. We should also create a static LeagueManager class that performs all the logic in the league.



The LeagueManager exposes only one public method – HandleInput.



The AddTeam, AddMatch, etc methods are all private.

They perform the following logic:

* AddTeam – creates a team and adds it to the League’s teams.
  + If the team exists, throw an exception with a descriptive message
  + Else, display a confirmation message
* AddMatch