Lab 2: Implementing Class Relations

Tània Pazos and Aniol Petit

Pompeu Fabra University

24292: Object Oriented Programming

October 26, 2023

Lab 2: Implementing Class Relations

The following paper describes the implementation and subsequent verification of the football application designed in Seminar 2. This lab puts the accent on implementing not only the classes forming the application but also the relations that can be established between them. Figure 1 illustrates the design that was taken as a reference to write the Java code.

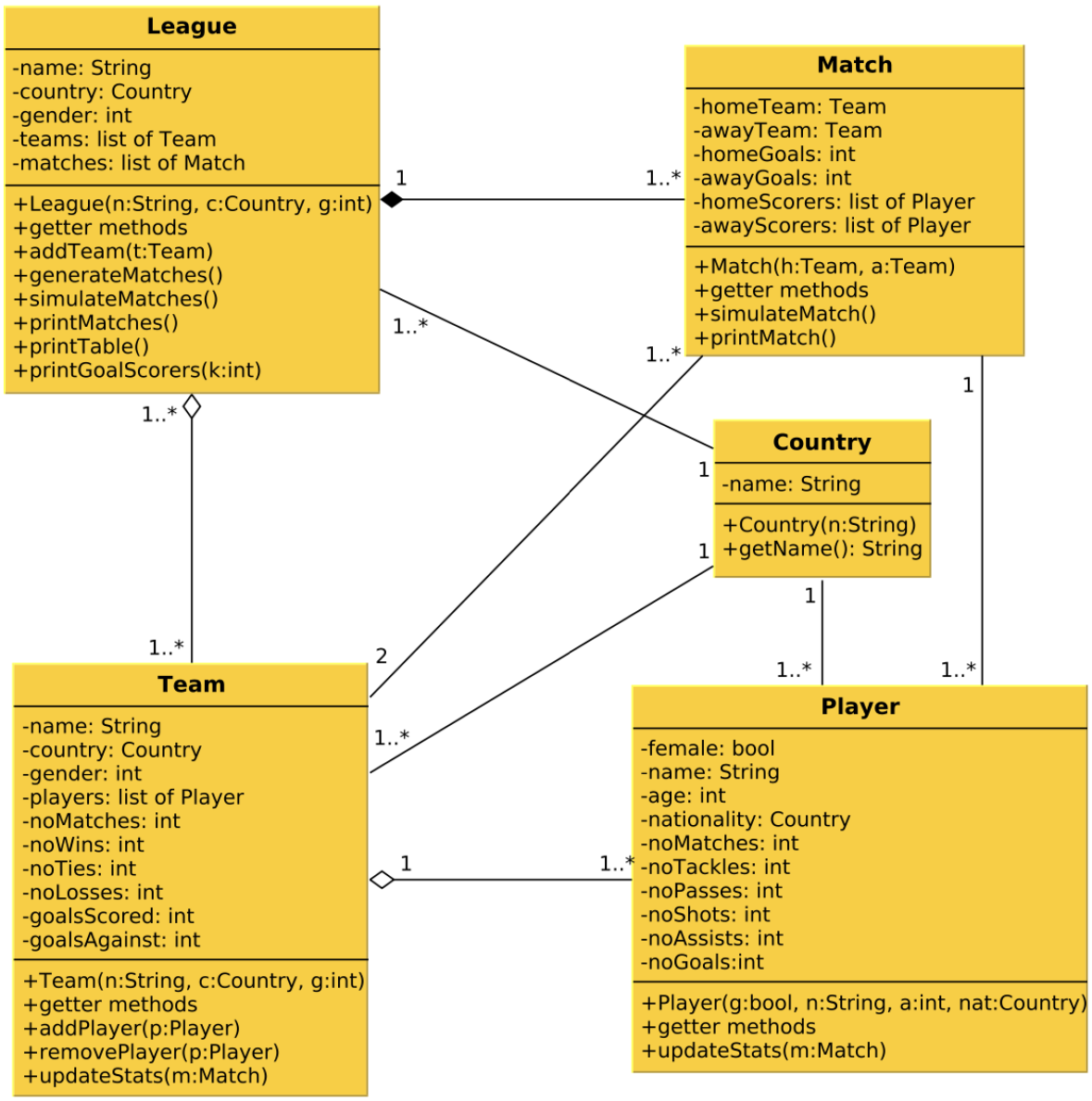
The Java program should be able to (i) specify all attributes and methods for the `Player`, `Team`, `Country`, `Match`, and `League` classes -making special attention to the fact that classes need attributes that correspond to the relations of type “composition”, “aggregation” and “association”, and (ii) define a test class with a main method, where instances of all classes are created and the various methods are implemented to ensure that there are no errors.

Firstly, the method `simulateMatch` of the `Match` class must generate a random number of goals for both the home and the away team and call the method `simulateScores` to add these goals to the attributes `homeScorers` and `awayScorers`. Indeed, the method `simulateScores` should randomly pick a player from the team and add them to the list of scorers. Once simulated, the method `printMatch` should print the names of the teams and the goals scored by each team.

Regarding the `League` class, the method generateMatches() should iterate through all pairs of teams in a double loop. As a consequence, exactly two matches must be created for each pair of teams -one home and one away- and the resulting matches should be added to the list of matches. The method `simulateMatches` should parse through the list of matches and call the `Match` method `simulateMatch` for each of them. The method `simulateMatches` should also call the `Team` method `updateStats` in order to update the statistics for both the home and the away teams that played in that match. Certainly, the method `updateStats` of the `Team` class and the `Player` class should reuse the code of Lab 1 and call the methods `playMatch` and `update`, as the information needed is contained in the Match class.

Figure 1

Football application design.



Solution Implementation

`Match` class

The attributes definition follow the design proposed in Figure 1.

As for the methods, the constructor `Match` correctly sets the value for the attributes home and away, both of them of type Team. The rest of the attributes are not included in the constructor method as it does not make sense to provide default values for statistics of home and away goals, and the lists of goalscorers will have added players later when simulating matches, but it also has no sense to assign some by default. There are also correctly implemented all the getter methods for every attribute in the class. The method `simulateMatch` creates to random integers and assign them to `homeGoals` and `awayGoals` attributes, to set the score of the match; and then it calls the `simulateScores` method, which given the number of goals of the home and away team, adds the goal scorers to their respective attributes. In the first described method we can point out that the amount of goals of both teams is limited to 10, to avoid results being very far from reality. About the `simulateScores` method, it is remarkable that when implementing it we got an out of bound error, since we generated a random number to determine the goal scorers, and that number could be out of the array; to fix that we just had to set the limit of the random number to the length of the array of players. Furthermore, we only allow each player to be added once to the array of goal scorers of the match, even if it could have scored several goals. Finally, the `printMatch` method just displays the result of the match.

`League` class

The attributes definition follow the design proposed in Figure 1.

As for the methods, the constructor `League` works similar to the `Match` one: it correctly sets the values for the attributes `name`, `country` and `gende`r, but the lists of matches and teams are to be completed.There are also getter methods for the attributes created in the constructor. The `addTeam` method adds a team to the list of teams in the

league if the given team matches the league gender, otherwise it displays a message stating that team can not be added due to its gender. Then there is the `generateMatches` method, which basically iterates over every pair of teams in the league and creates two matches between them (in one team 1 would be home and team 2 away, and in the other it would be the other way). Once the matches are created they are added to the list of matches of the league by calling the `addMatch` method, which just adds matches to the list. When implementing the `generateMathes` method, it was important to discard the pairs of teams where both teams are the same, as it is obvious that a team can not face itself. Finally, the method `simulateMatches` iterates through every match in the league's list of matches and calls the `Match` class method `simulateMatch` to simulate it and then updates both the home and away team statistics with the `updateStats` method from the `Team` class. With the method `printMatches`, we can print the results of all matches of the league. It iterates through every match of the league and calls the `Match` class method `printMatch`.

Possible Alternative Solutions

The only alternative solution we thought of was, in the simulateScores method of the Match class, setting the limit number of the index that will tell the scoring players to a certain value, e.g. 11, but that was not convincing as a team can have several players, and even if the number could be quite large, it would not ensure a free-of-errors program, so we decided to put the array length to avoid problems.

Conclusion

Multiple tests were executed to make sure that the program was implemented correctly. In the file `testPlayer.java`, two countries, four players, and two teams -with two players in each- were created, as shown in Figure 2. Then, the methods `playMatch` and `printStats` were called for `team1`, while for `player1` we called the methods `update` and `printStats`. A league "LaLiga" was created and the two teams were added to it, and the methods `generateMatches`, `simulateMatches`, and `printMatches`

emulated the start of the league. Finally, the method `printStats` was called for `team1` to depict the final statistics of the team and confirm the correct implementation of the league. Figure 3 shows the rest of the code in `testPlayer.java`, while Figure 4 captures the obtained output after execution.

Indeed, two matches are played between teams “Girona” and “Manchester” -one home and one away- and a random score is attributed to such match. What is more, the list of goal scorers of each time is randomly filled with players only if there were goals from that team. Finally, it can be observed that the statistics for team “Girona” are updated as expected, as the number of matches winned, lost and tied, and the goals scored and against are corrected according to the matches results.

All in all, the tests executed confirm that the program satisfies the expectations mentioned in the introduction of this paper.

Figure 2

Creation of instances of Country, Player, and Team classes.

```
// Create countries
Country country1 = new Country(n:"Spain");
Country country2 = new Country(n:"England");

// Create players
Player player1 = new Player(g:0, n:"Pol", a:20, country1);
Player player2 = new Player(g:0, n:"Aniol", a:19, country2);
Player player3 = new Player(g:0, n:"Jan", a:22, country1);
Player player4 = new Player(g:0, n:"Gerard", a:18, country2);

// Create a team
Team team1 = new Team(n:"Girona", country1, g:0);
Team team2 = new Team(n:"Manchester", country2, g:0);

// Add players to teams
team1.addPlayer(player1);
team1.addPlayer(player2);
team2.addPlayer(player3);
team2.addPlayer(player4);
```

Figure 3

Code for the league simulation.

```
// Play matches
team1.playMatch(favour:2, against:1);
team1.playMatch(favour:3, against:3);

//Print team stats
team1.printStats();

// Update and print player stats
player1.update(t:5, p:10, s:2, a:3, g:1);
player1.printStats();

//Create league and add teams
League league = new League(n:"LaLiga", country1, g:0);
league.addTeam(team2);
league.addTeam(team1);

//Generate and play matches
league.generateMatches();
league.simulateMatches();
league.printMatches();

team1.printStats();
```

Figure 4

Obtained output after execution.

```
Team Girona statistics:
Players in the team: Pol, Aniol
Matches Played: 2
Wins: 1
Ties: 1
Losses: 0
Points: 4
Goals Scored: 5
Goals Against: 4
Player Name: Pol
Matches: 1
Tackles: 5
Passes: 10
Assists: 3
Goals: 1
Manchester-Girona: 0-7
Manchester goal scorers:                Girona goal scorers: Aniol, Pol
Girona-Manchester: 0-0
Girona goal scorers:            Manchester goal scorers:
Team Girona statistics:
Players in the team: Pol, Aniol
Matches Played: 4
Wins: 2
Ties: 2
Losses: 0
Points: 8
Goals Scored: 12
Goals Against: 4
```