Lab 4: Implementing Interfaces

Tània Pazos and Aniol Petit

Pompeu Fabra University

24292: Object Oriented Programming

November 22, 2023

Lab 4: Implementing Interfaces

The aim of this paper is to create a mechanism for sorting league tables and lists of goals scorers as well as implementing the Java classes of the football application designed in Seminar 4. In particular, the focus of this project is to correctly implement the interface Comparable. Figure 1 shows the design for teams, team statistics, players, and players statistics, excluding all attributes and methods implemented in previous versions.

The Java program should be able to (i) implement the classes GoalkeeperStats, OutfielderStats, and TeamStats, (ii) implement the interface Comparable, (iii) implement the abstract class PlayerStats, and (iv) execute several tests by calling methods from the main method to ensure that the code meets all requirements.

Firstly, all attributes related to statistics and methods updateStats and printStats from Team should be moved to TeamStats; from Goalkeeper to GoalkeeperStats, and from Outfielder to OutfielderStats.

Secondly, classes TeamStats and OutfielderStats should implement the interface Comparable. Then, these two classes should override the method compareTo in order to sort teams and goal scorers (outfielders). In a league, the criteria to sort teams are as follows:

1. More points, defined as 3 * noWins + noTies.
2. Greater goal difference, defined as goalsFor - goalsAgainst.
3. More goals scored, i.e. goalsFor.

On the other hand, to determine if one outfielder has scored more goals than another the value of the attribute noGoals in the OutfielderStats should be compared -again, this must be done by overriding the method compareTo.

Thirdly, classes Team and Player should incorporate a dictionary stats using the existing class HashMap to identify the TeamStats and PlayerStats given a specific Competition. In addition, the method update of Team should update the team statistics after playing a match in a given competition by looking in the dictionary to see if the team already has any statistics linked to that competition. If not, a new instance of TeamStats must be

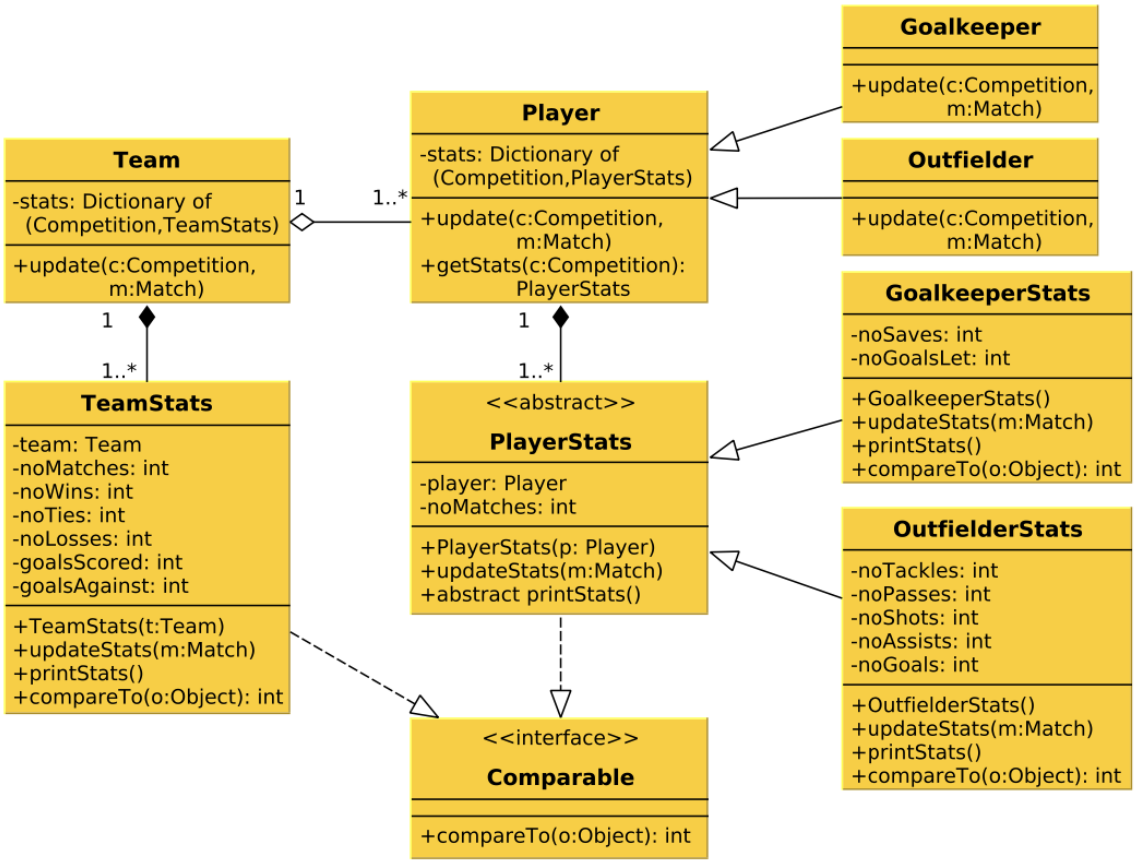
created and added to the dictionary for that competition. Then, the method updateStats of TeamStats should be called with the given match.

Similarly, the method update of Player should also update the player statistics after playing a match in a given competition by looking in the dictionary to check whether the player already has some statistics associated with that competition. If not, a new instance of PlayerStats should be created and added to the dictionary. Finally, the method updateStats of PlayerStats must be called with the given match. It should be noted that classes Goalkeeper and Outfielder should override the method update -since the method must be able to create an instance of Goalkeeper if the player is a goalkeeper, and an instance of Outfielder if the player is an outfielder- and updateStats, since goalkeepers and outfielders have different statistics.

Lastly, League should define the method printTable by creating a list of TeamStats, filling it with the team statistics of all teams in a specific competition, and sorting the list. Class Competition should implement the method printGoalScorers by implementing a method getOutfielderStats, which returns a list of OutfielderStats for all outfielders of all teams in a competition, sorting the list and printing the first k players using the method printStats of OutfielderStats.

Figure 1

Football application design



Solution Implementation

Team and player statistics

As previously mentioned, all attributes related to statistics were moved from Team to TeamStats, from Goalkeeper to GoalkeeperStats, and from Outfielder to OutfielderStats.

Sorting league tables

Following the criteria stated in the introduction, the overridden method compareTo in TeamStats was implemented as follows. Firstly, the number of points between two team statistics is compared. Since we want the teams to be sorted in descending order, if the current team has more points than the other team, the method returns -1 i.e. the current instance of TeamStats should be sorted before the other; if the current team has less points than the other, the method returns 1 i.e. the current team must be sorted after the other. If they have the same number of points, the goal difference, defined as goalsFor - goalsAgainst is compared following the same logic as the previous comparison. Again, if the goal difference is the same for both teams, the number of goals scored is compared. If the number of goals is also the same for both teams, the method returns 0, meaning the teams are equal.

Sorting goal scorers

In order to sort goal scorers, only one comparison is needed. Hence, in the overridden method compareTo, if the number of goals from the current outfielder is greater than the number of goals of another outfielder, the method returns -1 i.e. the current outfielder should be sorted before the other; if the number of goals are equal, the method returns 0; otherwise, the method returns 1.

Implementing a dictionary

The attribute type HashMap<Competition, TeamStats> stats included in Team allows for storing and retrieving the team statistics for a given competition. On the other hand, the method update of Team takes two parameters: a Competition and a Match. The method first checks whether there are existing statistics for the team in the given competition i.e., if there is a key-value pair in the dictionary stats. If not, it creates a new instance of TeamStats for

the current team, updates the information with the match passed as parameter with the function updateStats and creates the entry using .put with that key-value pair (Competition, TeamStats). Otherwise, i.e. if there is already an entry in the dictionary, the TeamStatistics stored are retrieved using .get, updated using updateStats, and stored again in the dictionary with .put. Finally, the statistics of the players forming the team are updated.

Similarly, class Player contains an attribute type HashMap<Competition, PlayerStats> stats to store the player statistics for a particular competition. In this case, however, the method update of Player is left empty so that Outfielder and Goalkeeper can override it: indeed, the method update will have to create an instance of Outfielder if the player is an Outfielder and an instance of Goalkeeper if the player is a Goalkeeper. What is more, the method updateStats called inside update has a different implementation for each type of player, since Outfielders and Goalkeepers have different statistics. Thus, in the Outfielder class, update checks if there is a value i.e. a PlayerStats for the Competition passed as parameter; if there is no value, a new instance of OutfielderStats is created with the current Outfielde, updated using the method updateStats of OutfielderStats and stored in the dictionary. Otherwise, the value of the PlayerStats is retrieved from the dictionary, updated and put back into the dictionary. Goalkeeper defines the method update exactly in the same way, but generating instances of GoalkeeperStats and calling the method updateStats of the class GoalkeeperStats.

Printing league tables and goal scorers

The method printTable of League begins by creating a list totalTeamStats to include the statistical information of all teams. Then, it iterates through the teams, retrieves its TeamStats and adds them to the list. The list is sorted using the method Collections.sort, which will sort the teams according to the criteria implemented in the overridden method compareTo of TeamStats. The method then prints a header for the table, with statistics like the number of matches played, wins, ties, losses, goals scored, goals against,and total points. To get the actual values, it iterates through all the TeamStats in the descending-order

sorted list TotalTeamStats and prints a line for each team with their statistics, thus creating a table.

On the other hand, the method printGoalScorers of the class Competition shows the k top goal scorers of the competition. Similarly to the method printTable from League, the method printGoalScorers creates a list totalOutfielderStats to store the OutfielderStats of all outfielders from a competition. Subsequently, it iterates through all the players of all teams in the competition and, after checking if the player is an outfielder with instanceof, retrieves the OutfielderSats associated with that competition in the dictionary and adds them to the list totalOutfielderStats. The list is then sorted using Collections.sort based on the criteria in the overridden compareTo method of the OutfielderStats class. Finally, using a loop that runs for the indicated number of goal scorers, the OutfielderStats for every outfielder are printed in a new line, calling the method printStats of OutfilederStats, that displays the player's name along with the number of matches played, goals, assists, shots, passes, and tackles.

Use case

In the file Test.java we tested all the new functionalities implemented in this lab. Hence, we printed the table and the top goal scorers for two different competitions: a league and a group play, to make sure that the methods were properly implemented in both types of competitions. We did not try the new functionalities for the Cup, since there is no table to print in this type of Competition. In addition, the method to print goalscorers is defined in Competition; hence, it works exactly the same for every subclass.

In both cases, the output shows everything we expected: all the matches and their results (feature that was already implemented in the previous lab), the table with the sorted teams based on their points and, in case of a draw, based on their goal difference and goals scored; and the desired top of goal scorers showing all the statistics of these players. Figure 2 shows the output result when testing the group play. Note the top goal scorers properly printed with their respective statistics and the table of each group, with the teams sorted by the defined criteria. Figure 3 displays the output for the case where the competition is a League, and we can see the same expected correct results.

Figure 2

Group play test

```
TOP 5 GOALSCORERS

               PLAYED GOALS   ASSISTS SHOTS   PASSES  TACKLES
Pau            4      16      3       26      14      16

               PLAYED GOALS   ASSISTS SHOTS   PASSES  TACKLES
Stuani         4      15      1       19      24      13

               PLAYED GOALS   ASSISTS SHOTS   PASSES  TACKLES
Pere           4      15      0       11      23      12

               PLAYED GOALS   ASSISTS SHOTS   PASSES  TACKLES
Aniol          4      13      2       16      17      7

               PLAYED GOALS   ASSISTS SHOTS   PASSES  TACKLES
Jan            4      13      2       14      16      28


Group 1 table:

               PLAYED  WINS    TIES    LOSSES  FOR     AGAINST POINTS
Osasuna        4       3       1       0       20      12      10
Atletic        4       1       2       1       15      15      5
Rayo           4       0       1       3       15      23      1


Group 2 table:

               PLAYED  WINS    TIES    LOSSES  FOR     AGAINST POINTS
Real           4       3       0       1       26      20      9
Girona         4       2       0       2       22      21      6
Madrid         4       1       0       3       17      24      3
```

```
Group 3 table:

               PLAYED  WINS    TIES    LOSSES  FOR     AGAINST POINTS
Lleida         4       3       1       0       28      11      10
Tarrago        4       1       1       2       20      23      4
Bcn            4       1       0       3       18      32      3
```

Figure 3

League test

```
                    League table

             PLAYED  WINS    TIES    LOSSES  FOR     AGAINST POINTS

Madrid       16      9       2       5       87      76      29
Atletic      16      8       1       7       65      74      25
Bcn          16      7       3       6       86      68      24
Lleida       16      8       0       8       70      78      24
Tarrago      16      6       4       6       83      71      22
Osasuna      16      7       1       8       63      55      22
Girona       16      6       3       7       52      73      21
Real         16      5       4       7       63      68      19
Rayo         16      5       4       7       67      73      19


TOP 5 GOALSCORERS

              PLAYED GOALS   ASSISTS SHOTS   PASSES  TACKLES
Gerard       16      50      11      57      77      85

              PLAYED GOALS   ASSISTS SHOTS   PASSES  TACKLES
Arnau        16      47      6       78      62      68

              PLAYED GOALS   ASSISTS SHOTS   PASSES  TACKLES
Pol          16      43      9       79      94      76

              PLAYED GOALS   ASSISTS SHOTS   PASSES  TACKLES
Aleix        16      43      7       95      92      83

              PLAYED GOALS   ASSISTS SHOTS   PASSES  TACKLES
Eric         16      39      9       80      70      85
```

Before concluding, it is worth mentioning that several issues were encountered with the implementation of the printTables method of the GroupPlay class. We wanted to reuse the printTablemethod of the League class to print the table for each group as every group is indeed a league, but when executing the program an error appeared stating that the stats were not properly being created for the teams, while when that was manually checked in the main it did not happen. As we could not find a reason nor a solution to this problem, we decided to implement manually the code to print the table for each group in the group play and it finally worked as desired.