Lab 3: Implementing Inheritance and Polymorphism

Tània Pazos and Aniol Petit

Pompeu Fabra University

24292: Object Oriented Programming

November 12, 2023

Lab 3: Implementing Inheritance and Polymorphism

The following paper describes the implementation of the football application designed in Seminar 3. Similarly to the previous lab, class relations play a key role in the application, particularly inheritance. Figure 1 shows the class diagram followed in the program.

The Java program should be able to (i) redefine the necessary classes from the previous lab in order to implement the new design, (ii) define new classes such as `NationalTeam`, `CupMatch`, `Goalkeeper`, `Outfielder`, `Competition`, `GroupPlay`, and `Cup`, and (iii) execute several tests by calling methods from the main method to ensure that the code meets all requirements.

Firstly, `NationalTeam` should override the method `addPlayer` from `Team` so that, every time a player is added, the code checks if the player has the correct nationality.

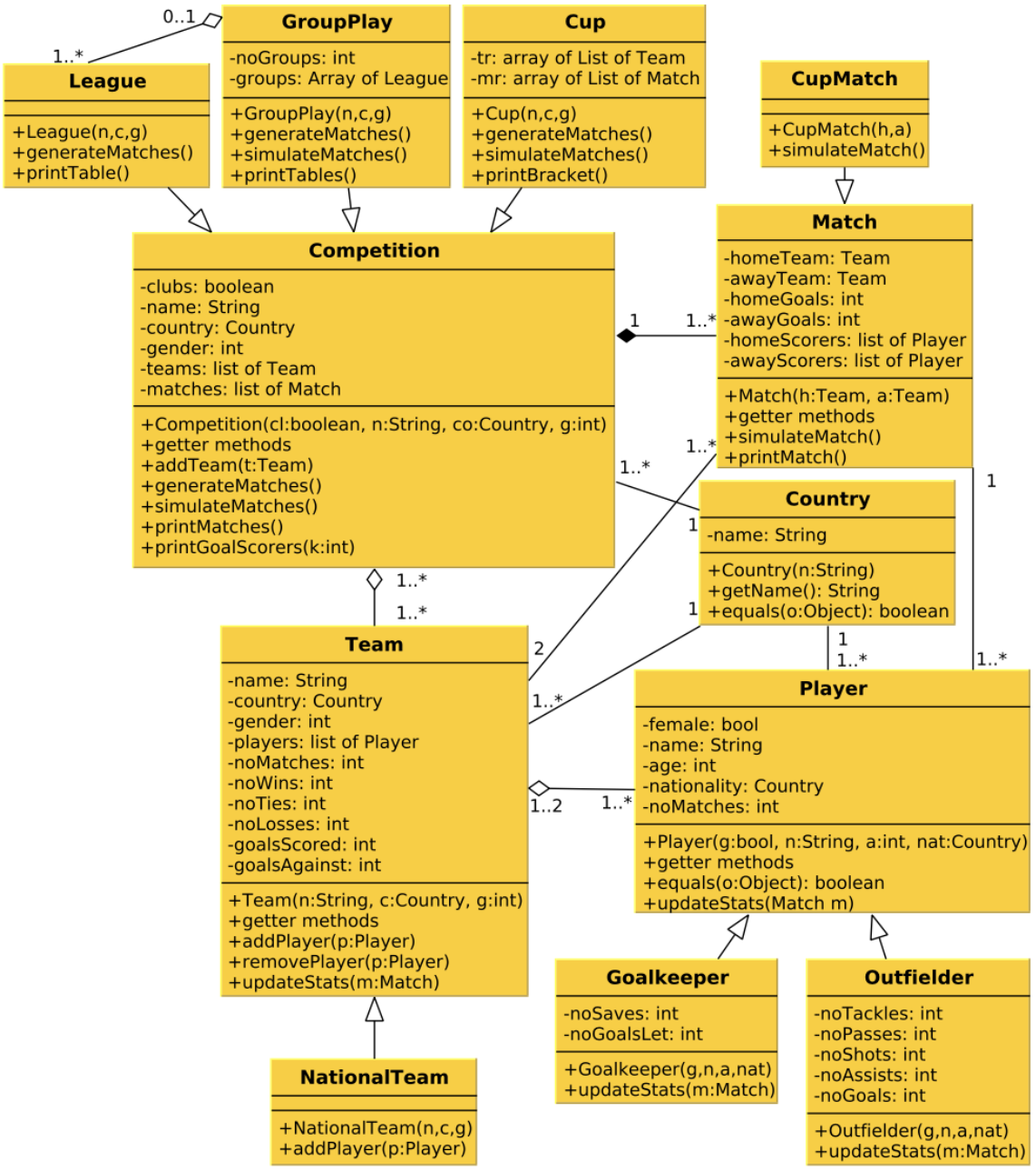
Secondly, the class `CupMatch` must override the method `simulateMatch` from `Match` to make sure that a match does not end in a tie. Thus, in the case of a tie, the match simulation continues with extra time and, if it is a draw again, with shoot-outs of five penalties. Of course, goal scorers for both the extra time and the penalties should be generated.

Classes `Goalkeeper` and `Outfielder` extends from the `Player` class. `Outfielder` contains the previously defined statistics of the `Player`, whereas `Goalkeeper` defines its own statistics.

Finally, a new parent class `Competition` must be defined by reusing attributes and methods from the `League` class implementation in Lab 2. This new class is able to distinguish between club competitions and international competitions using the boolean clubs. Consequently, the method `addTeam` should only allow club teams in club competitions and national teams in international competitions.

Figure 1

Football application design



As mentioned above, three classes inherit from `Competition`: `League`, `Cup`, and `GroupPlay`. On the one hand, `League` now contains the method `generateMatches`, that specifically generates matches for a league. Class `Cup` must be able to generate and simulate matches for several rounds, until only one winner is selected. To do so, lists of teams and lists of matches of each round should be stored in the arrays `tr` and `mr` respectively. In the first round, teams must be chosen randomly using the method `Collections.shuffle`, while in subsequent rounds only the winner teams play. Finally, the `GroupPlay` class should maintain an array of `League` so that each group is an instance of `League`.

Solution Implementation

Class `NationalTeam`

`NationalTeam` extends from `Team` and overrides the method `addPlayer`. To check if two countries are equal, i.e., they have the same name, the code overrides the method `equals` that `Country` inherits from `Object`.

Class `CupMatch`

`CupMatch` extends from `Match` and overrides the method `simulateMatch` so that no match ends in a tie. Indeed, the method introduces features like extra time and penalty shootouts to select a winner. During extra time, the method generates a random number of goals and adds them to the original match goals. If there is no clear winner after the extra time, a random number of goals for the penalty shootouts is generated and added to the total number of goals -from both the match and the extra time-. This process is repeated until there is a different number of goals for each team.

Classes `Goalkeeper` and `Outfielder`

As stated in the introduction, both `Goalkeeper` and `Outfielder` extend from `Player`. On the one hand, `Goalkeeper` introduces two new statistics, i.e., two new attributes, that only goalkeepers have: `noSaves` and `noGoalsLet`. Then, the method

`uupdateStats` generates a random number to update these two statistics and adds one to the number of matches of the player.

Similarly, `Outfielder` stores the statistics that `Player` had in Lab 2 and updates the statistics of the player by generating random numbers for all statistics and incrementing by one the number of matches played.

Implementing competitions

Class `Competition` simply modifies the method `addTeam` of the Lab 2 version of `League` class. The new version of `addTeam` enforces that the added team and the competition have gender and club type compatibility. If the conditions are not met, error messages are displayed. The method `generateMatches` is left empty, as each type of competition will define its own way of generating matches.

`League` extends from `Competition` and simply overrides the method `generateMatches` reusing the code of Lab 2.

`Cup` incorporates two arrays of LinkedLists `tr` and `mr` as attributes, which store the teams and matches of each round. In addition, it introduces an attribute of type `Team spareteam` to store, in case there is an odd number of teams in the cup, the team that is not paired in each round. The method `simulateMatches` takes as parameter a list of `CupMatch` objects that represent matches of a round. Each match is simulated, the statistics of each team are updated, the match details are printed, and the winner is added to a LinkedList that contains the teams that will play in the next round. If there is a spare team, the spare team is also added to the next round. The list of the teams for the next round are added to the array `tr`. The method `generateMaches` generates the matches of a round. First, it creates random pairs of teams using `Collections.shuffle`, and matches are established by pairing consecutive teams in the list `roundTeams`. In the case of an odd number of teams, the last team is stored in the attribute `spareTeam`. Then, the method adds the matches generated to the array `mr`, prints the round number, and calls the method

`simulateMatches` to select winners and hence determine the next round. The process does not end until only one team -the winner- appears in the last position of the `tr` array.

Finally, the class `GroupPlay` defines methods to split teams into groups and, similarly to the other two classes, generate matches specifically for group play. The method `formGroups` creates the number of groups set in the attribute `noGroups` by shuffling the teams of each league in the list of leagues passed as parameter. Then, teams are added to each league until the computed number of teams per group is reached. Should any remaining teams be left without a group, they are added to the last league. The method `generateMacthes` creates a match between every two teams within a group and adds the match to the list of matches. Lastly, the method `addLeague` adds a league to the list of groups after checking that the gender and club condition of the league are the same as those of the group play.

Possible Alternative Solutions

Initially, the implementation of the class `Cup` did not include the attributes `tr`, `mr`, and `spareTeam`. The method `generateMatches` directly shuffled and paired the teams and, in case of an odd number of teams, the spare team was not paired.

On the other hand, the method `simulateMatches` eliminated the losing teams until there was a winner. Hence, the information regarding the teams and matches of each round was lost.

Even though we ended up discarding this implementation because it did not include the spare team in the cup, it was a simple and valid design. Figure 2 shows the code of this alternative implementation of the `Cup` class.

Conclusion

Multiple tests were executed to make sure that the program was implemented correctly. This report will explain the results displayed in Figure 3, which were obtained after testing the `Cup` class. In the file `Test.java`, nine teams are created -Girona, Bcn, Lleida, Tarrago, Madrid, Athletic, Osasuna, Rayo, and Real- and a player is added to each team.

Then, these nine teams -we picked an odd number on purpose to check how the spare team was handled- are grouped into a cup and the cup matches are generated. After executing, for Round 1 the winners are Bcn, Real, Madrid, and Girona, and the spare team is Osasuna; all these winner teams play in Round 2 except for Madrid, which is now the spare team, and the new winners are Osasuna and Bcn; in Round 3, Madrid wins and Osasuna is the spare team; finally in Round 4 a match is played between Madrid and Osasuna, and Osasuna is declared winner team. In a nutshell, the program does meet the criteria established in the introduction of this report.

Figure 2

Alternative implementation of the `Cup` class

```java
public class Cup extends Competition{
  public Cup(boolean cl, String n, Country c, int g){
    super(cl, n, c, g);
  }

  public void generateMatches(){
    Collections.shuffle(teams); //randomize teams
    int size = teams.size();
    if(size % 2 == 0){ //case even number of teams
      for(int i = 0; i < size; i+=2){ //increase i by 2 because we pair 2
consecutive teams at a time
        CupMatch match = new CupMatch(teams.get(i), teams.get(i+1));
        addMatch(match);
    }
    }
    else{ //case odd number of teams
     for(int i = 0; i < size - 1; i+=2){ //we iterate until the penultimate team
because this one doesn't have a pair
        CupMatch match = new CupMatch(teams.get(i), teams.get(i+1));
        addMatch(match);
      }
    }
  }

  public void simulateMatches(){
    generateMatches(); //generate the matches for the first round so the
simulation can start
    int i = 1;
```

```
    while(teams.size() > 1){
      System.out.println("\nRound " + i);
      for(Match match : matches){
        match.simulateMatch();
        match.getHome().updateStats(match);
        match.getAway().updateStats(match);
        match.printMatch();
        if(match.gethomeGoals() > match.getawayGoals()){
          teams.remove(match.getAway());
        }
        else{
          teams.remove(match.getHome());
        }
      }
      matches.clear(); //We don't want the matches of the first round to be
simulated again
      generateMatches();
      i++;
    }
    System.out.println("Winner team is " + teams.get(0).getName());

  }
}
```

Figure 3

Terminal output to test the `Cup` class

```
Round 1
Tarrago-Bcn: 3-5
Tarrago goal scorers: Pau               Bcn goal scorers: Pol
Rayo-Real: 6-11
Rayo goal scorers: Ferran               Real goal scorers: Pere
Atletic-Madrid: 6-9
Atletic goal scorers: Marc              Madrid goal scorers: Gerard
Lleida-Girona: 4-6
Lleida goal scorers: Jan                Girona goal scorers: Aniol

Round 2
Osasuna-Real: 9-8
Osasuna goal scorers: Roger             Real goal scorers: Pere
Girona-Bcn: 3-6
Girona goal scorers: Aniol              Bcn goal scorers: Pol

Round 3
Bcn-Madrid: 2-8
Bcn goal scorers: Pol           Madrid goal scorers: Gerard

Round 4
Osasuna-Madrid: 6-1
Osasuna goal scorers: Roger             Madrid goal scorers: Gerard
Winner team is Osasuna
```