Lab Session 4: Implementing interfaces
24292-Object Oriented Programming

1 Introduction

In this lab session the aim is to implement a mechanism for sorting league tables and lists of goal scorers. The lab session consists in implementing the Java classes that were designed in Seminar 4, paying special attention to the implementation of the interface `Comparable`. Recall that in Java, the keyword `implements` is used to indicate that a class implements an interface.

On the next page you will find a sample design for teams, team statistics, players, and player statistics, which you can use instead of your own design.

2 Team and player statistics

First implement the classes related to team statistics and player statistics. Note that you have to move some attributes from `Team` to `TeamStats`, from `Goalkeeper` to `GoalkeeperStats`, and from `Outfielder` to `OutfielderStats`. Likewise, some of the methods in these classes have to be moved (`updateStats` and `printStats`).

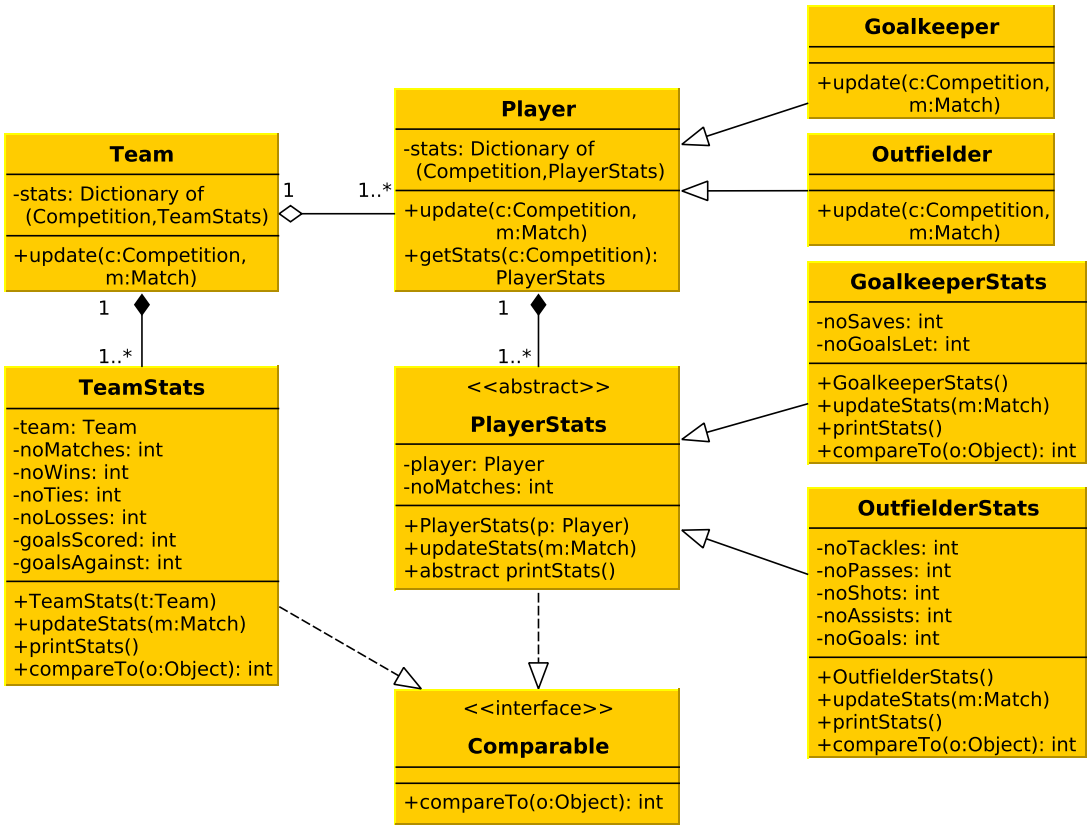
2.1 Sorting league tables

In a given league, the criteria for determining whether one team should be sorted before another team are as follows:

1. More *points*, defined as $3 * \texttt{noWins} + \texttt{noTies}$.
2. Better *goal difference*, defined as $\texttt{goalsFor} - \texttt{goalsAgainst}$.
3. More goals scored, i.e. `goalsFor`.

If two teams are equal according to all three criteria, they can be sorted arbitrarily.

In `TeamStats`, the idea is to override the method `compareTo` in order to correctly sort teams. Recall that `compareTo` should return -1 if the current instance of `TeamStats` should be sorted before another, 0 if they are equal, and 1 if the current instance should be sorted after another.


Goalkeeper
+update(c:Competition, m:Match)
Player
-stats: Dictionary of (Competition,PlayerStats)
+update(c:Competition, m:Match)
+getStats(c:Competition): PlayerStats
Outfielder
+update(c:Competition, m:Match)
Team
-stats: Dictionary of (Competition,TeamStats)
+update(c:Competition, m:Match)
1
1..*
1
1..*
1
1..*
GoalkeeperStats
-noSaves: int
-noGoalsLet: int
+GoalkeeperStats()
+updateStats(m:Match)
+printStats()
+compareTo(o:Object): int
<<abstract>>
PlayerStats
-player: Player
-noMatches: int
+PlayerStats(p: Player)
+updateStats(m:Match)
+abstract printStats()
TeamStats
-team: Team
-noMatches: int
-noWins: int
-noTies: int
-noLosses: int
-goalsScored: int
-goalsAgainst: int
+TeamStats(t:Team)
+updateStats(m:Match)
+printStats()
+compareTo(o:Object): int
OutfielderStats
-noTackles: int
-noPasses: int
-noShots: int
-noAssists: int
-noGoals: int
+OutfielderStats()
+updateStats(m:Match)
+printStats()
+compareTo(o:Object): int
<<interface>>
Comparable
+compareTo(o:Object): int

2.2 Implementing Comparable

There are two ways to implement the interface `Comparable` in Java:

1. Implement `Comparable` and override `compareTo` as shown in the design:

   ```
   public class TeamStats implements Comparable {
       ...
       public int compareTo(Object o) {
       ...
       }
   }
   ```

   In this case it is necessary to perform a *downcast* to TeamStats before making the comparison.

2. Implement `Comparable<TeamStats>` and override `compareTo` as follows:

   ```
   public class TeamStats implements Comparable<TeamStats> {
       ...
       public int compareTo(TeamStats ts) {
       ...
       }
   }
   ```

   In this case no downcast is necessary.

2.3 Sorting goal scorers

Determining if one outfielder has scored more goals than another is easy: just compare the value of the attribute `noGoals` in the class `OutfielderStats`. This is simply done by overriding the method `compareTo` in `OutfielderStats`.

3 Implementing a dictionary

The next step is to modify the classes `Team` and `Player` in order to incorporate a dictionary from competitions to statistics. A dictionary can be implemented in Java using the existing class `HashMap`, which has to be imported in the class file as usual. Please have a look at the methods defined in `HashMap`, especially `put` and `get`. The dictionary has to be created in the constructor of `Team` and `Player`.

The method `update` of `Team` should update the team statistics as a result of playing a match in a given competition. This is done by first looking in the dictionary to see if the team has any statistics associated with the competition. If not, you should create a new instance of `TeamStats` and add it to the dictionary for the current competition. Then you can simply call the method

`updateStats` of `TeamStats` with the given match. As before, `update` should also call the `update` method of `Player` on all players in the team.

Likewise, the method `update` of `Player` should update the player statistics as a result of playing a match in a given competition. This is done by first looking in the dictionary to see if the player has any statistics associated with the competition. If not, you should create a new instance of `PlayerStats` and add it to the dictionary for the current competition. Note that the classes `Goalkeeper` and `Outfielder` override the method `update`. The reason is so that the method can create an instance of `GoalkeeperStats` if the player is a goalkeeper, and an instance of `OutfielderStats` if the player is an outfielder.

Once the player statistics for a competition exists in the dictionary, you can simply call the method `updateStats` of `PlayerStats` with the given match. As before, the implementation of `updateStats` is different for goalkeepers (in `GoalkeeperStats`) and outfielders (in `OutfielderStats`).

4 Printing league tables and goal scorers

The last step of the lab session is to print league tables and lists of goal scorers. In `League`, implement the method `printTable` by creating a list of `TeamStats` and filling it with the team statistics of all teams in the current competition (by calling the method `getStats` of `Team`). Then you can simply sort the list of `TeamStats` using the method `Collections.sort`. To print the table on a nice format, you can play with the method `printStats` of `TeamStats`.

Finally, implement the method `printGoalScorers` of the `Competition` class. To do so, it is a good idea to implement a method `getOutfielderStats`, which returns a list of `OutfielderStats` for all outfielder players of all teams in the competition. Once you have such a list, simply sort it using `Collections.sort` and print the first k players (using the method `printStats` of `OutfielderStats`).

5 Documentation

Apart from the source code, you should also hand in a document that outlines the solution of the problem. To elaborate the document you can use the following guidelines regarding the content:

1. An introduction where the problem is described. For example, what should the program do? Which classes do you have to define? Which methods do you have to implement for these classes?

2. A description of possible alternative solutions that were discussed, and a description of the chosen solution and the reason for choosing this solution rather than others. It is also a good idea to mention the related theoretical concepts of object-oriented programming that were applied as part of the solution.

3. A conclusion that describes how well the solution worked in practice, i.e. did the tests show that the classes were correctly implemented? You can also mention any difficulties during the implementation as well as any doubts you might have had.

The source code and documentation should all be uploaded to a directory `Lab4` of your Git repository **prior to the next lab session**.