Seminar 3: Designing an application using inheritance
24292-Object Oriented Programming

1 Introduction

The objective of this seminar is to learn how to design an application that uses the concept of inheritance and polymorphism. Apart from the relations, each class has a number of attributes and methods that are to be set to complete the design. Remember that each completed design has to be a connected graph.

The solution to these exercises will be implemented in Java during Lab session 3.

2 Review

As in the last seminar, we will use the relations seen in class. We will pay special attention to the **inheritance** relations and the **polymorphism** that we can achieve with them.

To indicate in the design that you want to **override** a method, simply repeat the method again in the appropriate subclass.

3 Description of the football application

We want to expand the football application so that it can represent and simulate different types of football competitions. There are three different types of competitions:

1. League, in which each team plays each other team twice (matches can be tied).
2. Cup, in which teams are eliminated when they lose, and several rounds are played until only one team remains (matches cannot be tied).
3. Group play, in which teams are divided into several groups, and only the top team(s) of each group qualify for the next event. In each group, teams can play each other either once or twice.

Each competition has a name (e.g. "La Liga", "Women's World Cup", etc.) Each competition involves a set of football teams, each of which consists of a set of players. Teams and players are defined as in Seminar 1. However, teams can be of two types: club teams or national teams. The players of a national team have to have the same nationality as the country of that team. Competitions can be either club competitions (between club teams) or international competitions (between national teams). As before, each competition also has an associated gender (male, female, or mixed), and the participating teams have to have the correct gender designation. It should be possible to add teams to a competition according to these rules.

Each competition involves a number of matches. Each match is played between exactly two teams, and the application should record the score (goals scored of each team) and the goal scorers (list of players of each team who scored the goals). As before, the application should be able to generate the associated matches, and then simulate the matches. However, for cups, the generated matches in one round depend on which teams won the matches of the previous round, so generation and simulation have to take place in stages. In addition, cup matches cannot be tied, so the teams have to keep playing until there is a winner (e.g. using extra time and penalties).

After a match is simulated, the statistics of the participating teams and players should be updated. It should be possible to print the result of all simulated matches. For leagues and group play, it should be possible to print the final tables of the leagues or groups. For cups, it should be possible to print the bracket (i.e. the tree of matches in which the winner always advances). For all types of competitions, it should be possible to print a list of the top k goal scorers.

In addition, we want to expand the application so that players can have different types. For now, we will just distinguish between goalkeepers and outfielders. Goalkeepers do not have the same statistics as outfielders; they record the number of games played, the number of saves, and the number of goals scored against. When a match is simulated, typically the goal scorer will not be a goalkeeper (or only with a very small probability).

4 Design

Given the specifications in the previous sections, the seminar consists in defining all the classes and relations of the football application. For each class you will have to indicate attributes and methods. You will also have to indicate the cardinality and concrete type of relation.

To be successful, the design needs to include mechanisms for interaction between classes, since typically the information needed to perform an operation is stored in instances of different classes. A good design should attempt to conform to the following standards:

1. A large amount of decoupling, i.e. try to include as few relationships between classes as possible without breaking the necessary interaction.

2. A class should not call methods of other classes than those it is directly related to, which means that multiple methods might be needed in case information has to propagate through several classes.