Lab 5: Implementing inheritance and polymorphism in C++

Tània Pazos and Aniol Petit

Pompeu Fabra University

24292: Object Oriented Programming

December 10, 2023

Lab 5: Implementing inheritance and polymorphism in C++

This report details the design and implementation of a football application in C++, with a focus on the creation and functionality of key classes integral to the application. Figure 1 shows the design diagram provided. The main goal of this project is to develop a system that encompasses various aspects of football, including matches, countries, teams, players, goalkeepers, and outfielders. The classes Match, Country, and Team, have been provided and implemented. The core responsibility lies in creating three additional classes: Player, Goalkeeper, and Outfielder. The abstract Player class contains all attributes and methods that subsequent subclasses must inherit and implement. Following C++ best practices, header files are used and include guards employed to prevent duplication.

On the one hand, the abstract Player class is encapsulated in the header file Player.hpp and uses protected visibility for attributes, since we intend to inherit from it. All attributes and methods from the Player class should be implemented in the program.

Subsequently, Goalkeeper and Outfielder classes are to be developed in their respective header files. These classes inherit from the Player class and hence must override specific methods such as updateStats and printStats. In order to implement inheritance, both classes should use the syntax "class Goalkeeper(...) : public Player(...)", i.e., call the Player's constructor on the same line as theirs with the appropriate arguments.

Finally, a main function in a separate code file (with extension .cpp) should be created for comprehensive testing. The main function should be able to:

- Create a few countries.
- Create a few outfielders from different countries.
- Create two teams that each include some outfielders.
- Create a match between the two teams, simulate the match and print the result.
- Update the statistics of each player as a result of playing the match.
- Print the statistics of each player.

This code file should include all header files, bearing in mind that inclusion is recursive; in other words, the test file must only include Goalkeeper.hpp and Outfielder h.pp, since those headers recursively include all other headers.

Solution implementation

Player class

The Player class serves as the abstract base class for the football application. It contains essential player attributes such as gender, name, age, nationality, and match statistics. The class employs protected visibility for attributes, ensuring access to the Outfielder and Goalkeeper classes that inherit from it. It should be noted that, in contrast to the previous labs, a pointer to the Country class represents the player's nationality. Two abstract methods, updateStats and printStats, are declared so that their implementation can be specified in the subclasses Outfielder and Goalkeeper. What is more, an implementation of the equality operator '==' is included to compare two players based on the attribute name. Indeed, without this operator implementation, direct equality comparison using '==' between players would not be permitted. The following code depicts the program implementation of the equality operator.

```
C/C++
 bool operator==(const Player& other) const {
      return name == other.name;
    }
```

Goalkeeper class

The Goalkeeper class inherits from the abstract Player class, using the expected syntax stated in the introduction.

```
C/C++
class Goalkeeper : public Player
```

It contains two private attributes noSaves and noGoalsLet, and initializes the inherited attributes through both its constructor and the Player class constructor using the following

code. Note that the arguments chosen to initialize the Goalkeeper attributes are consistent with the attributes declared in the superclass Player.

```
C/C++
Goalkeeper(bool g, std::string n, int a, Country * c) : Player(g, n, a, c){}
```

The updateStats method is implemented to calculate the number of goals let in a match, incorporating randomization for the number of saves. The method iterates through opposing team players, updating statistics accordingly. The printStats method displays relevant goalkeeper statistics, such as the number of matches played, saves made, and goals conceded.

Outfielder class

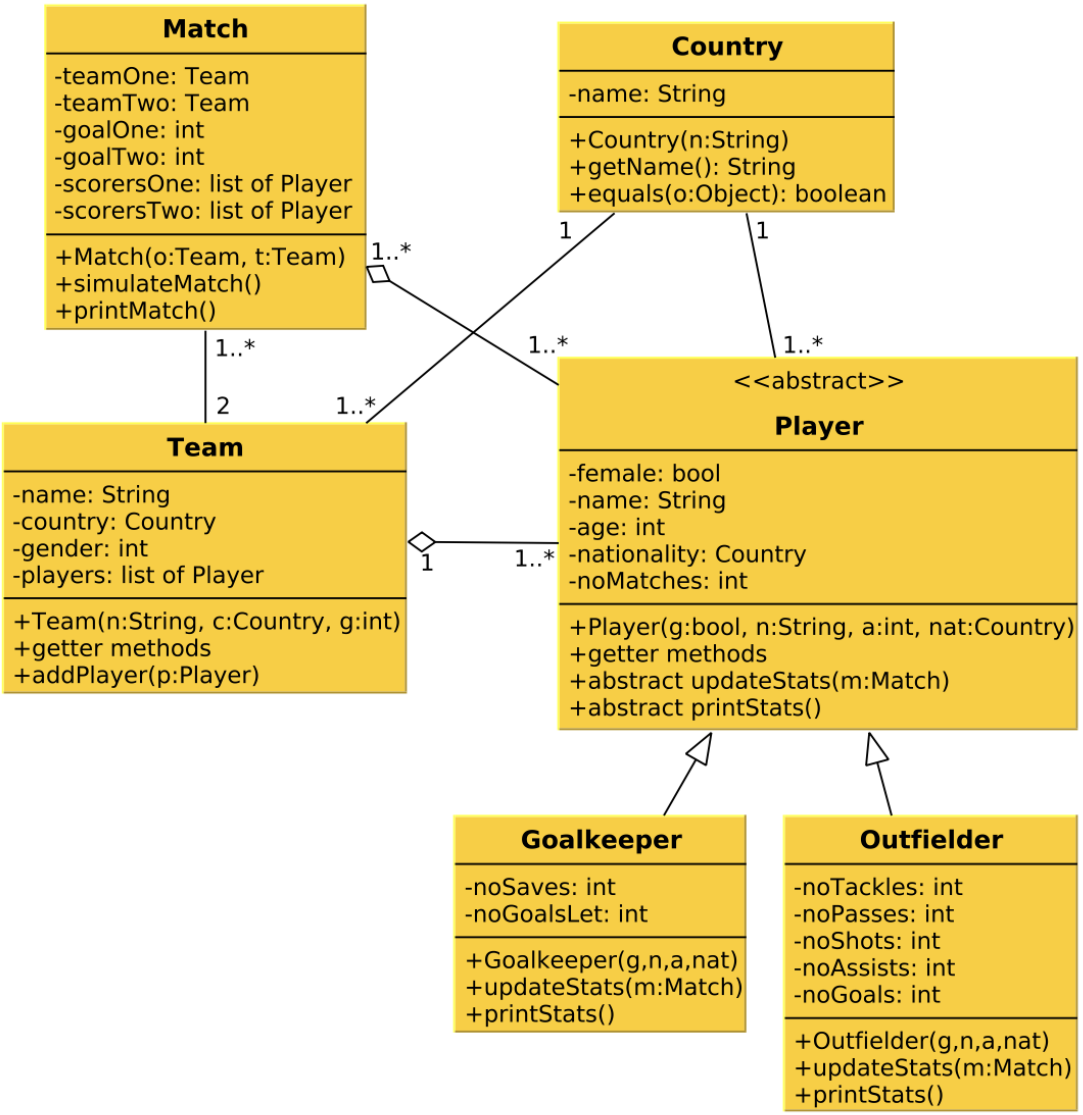
The Outfielder class, like Goalkeeper, inherits from the Player class. In a similar way, it contains private attributes representing all those statistics that are only valid for an Outfielder. Again, it initializes the inherited attributes by calling the constructor and the Player's constructor in the same line, using the syntax below.

```
C/C++
Outfielder(bool g, std::string n, int a, Country * c) : Player(g, n, a, c){}
```

Furthermore, the updateStats method is implemented to compute various statistics, including goals scored, tackles, passes, shots, assists, and the number of matches played. The printStats method displays comprehensive outfielder statistics, displaying the number of matches played as well as the number of tackles, passes, shots, assists, and goals.

Figure 1

Football application design



Test

After several tests, it can be concluded that the final implementation worked properly. It should be noted that in the provided code some considerations were not taken into account; for instance, goalkeepers were allowed to score goals, which could lead to unmatching results between the match results and the number of goals scored by each player (since goalkeepers would always have 0 goals scored, as they do not have an attribute noGoals). Apart from that, which we solved by just not adding goalkeepers to the teams, everything worked properly as it can be seen in Figure 2 and Figure 3.

Conclusion

The challenges encountered in implementing the football application primarily arose due to a lack of comprehensive understanding of the formal aspects of C++ programming. We found it difficult to structure classes, manage header files, and handle dependencies in C++, leading to extensive searches for relevant C++ documentation. This knowledge gap significantly extended the time spent on the development process.

Moreover, confusion emerged from our previous labs with Java programming. The differences in syntax between C++ and Java created uncertainty when translating concepts from one language to the other.

In a nutshell, the main hurdle we faced in developing the football application was the lack of understanding of formal C++ concepts, necessitating frequent reference to documentation.

Figure 2

Test output

```
Girona-Madriz: 5-5
Aleix STATS:
NUMBER OF MATCHES: 1
NUMBER OF TACKLES: 7
NUMBER OF PASSES: 7
NUMBER OF SHOTS: 10
NUMBER OF ASSISTS: 1
NUMBER OF GOALS: 1

Savio STATS:
NUMBER OF MATCHES: 1
NUMBER OF TACKLES: 7
NUMBER OF PASSES: 7
NUMBER OF SHOTS: 10
NUMBER OF ASSISTS: 1
NUMBER OF GOALS: 3

Tsygankov STATS:
NUMBER OF MATCHES: 1
NUMBER OF TACKLES: 7
NUMBER OF PASSES: 7
NUMBER OF SHOTS: 10
```

Figure 3

Test ouput

```
NUMBER OF ASSISTS: 1
NUMBER OF GOALS: 1

Joselu STATS:
NUMBER OF MATCHES: 1
NUMBER OF TACKLES: 7
NUMBER OF PASSES: 7
NUMBER OF SHOTS: 10
NUMBER OF ASSISTS: 1
NUMBER OF GOALS: 1

Vinicius STATS:
NUMBER OF MATCHES: 1
NUMBER OF TACKLES: 7
NUMBER OF PASSES: 7
NUMBER OF SHOTS: 10
NUMBER OF ASSISTS: 1
NUMBER OF GOALS: 1

Nico Paz STATS:
NUMBER OF MATCHES: 1
NUMBER OF TACKLES: 7
NUMBER OF PASSES: 7
NUMBER OF SHOTS: 10
NUMBER OF ASSISTS: 1
NUMBER OF GOALS: 3
```