LAB 5 - PROGRAMACIÓN ORIENTADA A OBJETOS

Implementación de herencia y polimorfismo en C++

El objetivo de este laboratorio es implementar una pequeña versión de la aplicación de fútbol, como la que hemos estado trabajando hasta ahora, en C++. En esta práctica el profesorado nos facilita un código en el que han sido implementadas las clases Match, Country y Team, por tanto quedaban por implementar las tres clases restantes: Player, Goalkeeper y Outfielder.

Para la implementación de la clase Player como vemos indicado en las clases Match y Team es preciso que esta aparezca en un archivo header, como Player.hpp. Tal como lo indica el informe del Lab 5 facilitado por el profesorado, hemos seguido la siguiente estructura para evitar la duplicaciones que facilitan los errores de compilación cuando dos o más clases incluyen el mismo archivo header.

```
#ifndef _PLAYER_
#define _PLAYER_
```

// código del archivo header Player.hpp

```
#endif
```

Por otro lado, para incluir la clase Country en el header de la clase Player no hay ningún problema, pero para incluir la clase Match nos encontramos con un ejemplo de dependencia cíclica ya que en la clase Match también ha sido añadida la clase Player. Para solucionarlo hemos incluido la clase Match en la clase Player de la siguiente forma:

```
#include "Country.hpp"
class Match;
```

Dado que C++ no tiene una palabra clave para definir una clase abstracta (como lo es Player ya que tiene dos métodos abstractos updateStats y printStats, consequentemente la clase Player también lo es) ha sido necesario usar la palabra clave virtual y establecer el puntero del método en 0:

```
virtual void updateStats(Match * m) = 0;
virtual void printStats() = 0;
```

Al implementar las clases Goalkeeper y Outfielder ha sido preciso incluir las clases Match y por supuesto Player en ambas clases. En ninguno de los casos se creaba una dependencia cíclica por tanto se han podido incluir las clases de forma estándar y sin problema. Aún y así en ambos casos para indicar la relación de herencia de las clases Goalkeeper y Outfielder que comparten con la clase Player, ha sido necesario señalarlo en la definición de la clases y de sus respectivos constructores.

```
class Goalkeeper : public Player {
```

```
public:
    Goalkeeper(bool g, std::string n, int a, Country * nat) : Player(g, n, a, nat) {
```

```
class Outfielder : public Player {
```

```
public:
    Outfielder(bool g, std::string n, int a, Country * nat) : Player(g, n, a, nat) {
```

En estas dos clases, para indicar que no son abstractas aunque hereden de una que sí lo es, hemos hecho un override de los métodos abstractos de la clase Player.

```
void updateStats(Match* m) override {
```

```
void printStats() override{
```

Finalmente, para implementar la función main que nos permitirá probar que realmente el código al ejecutarse hace la función correspondiente, nos hemos basado en los realizados en los Lab 2 y Lab 3. La implementación de la función main de esta práctica era más sencilla, ya que hemos creado dos country, cuatro outfielder de diferentes country, dos equipos que incluyen dos outfielders cada uno, un partido entre los dos equipos, que simula el partido e imprime el resultado y finalmente actualiza las estadísticas de cada jugador como resultado de jugar el partido e imprime las estadísticas de cada jugador.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

Match Result:
FC Barcelona-Barcelona Fem: 5-5
Player Statistics:

Stats Team FC Barcelona:
Lionel Messi--> Matches: 1, Tackles: 6, Passes: 7, Shots: 1, Assists: 2, Goals: 4
Karim Benzema--> Matches: 1, Tackles: 6, Passes: 2, Shots: 3, Assists: 0, Goals: 1

Stats team Barcelona Fem:
Ingrid Engen--> Matches: 1, Tackles: 2, Passes: 4, Shots: 2, Assists: 0, Goals: 3
Irene Paredes--> Matches: 1, Tackles: 5, Passes: 2, Shots: 1, Assists: 2, Goals: 2
PS C:\Users\julia\UNIVERSIDAD\java files\lab5>
```

Al principio cuando ejecutamos el programa después de realizar el main, todos los stats parecían resultar 0 y al buscar el error nos dimos cuenta que los stats no se actualizaban correctamente. Una vez detectado y solucionado este problema, el código ya funcionó con normalidad.

Durante el proceso, hemos ido compilando frecuentemente para la comprobación de errores.