INFORME LAB 5, DE JAVA A C++

Introducción: En esta sección, se describe el problema. Por ejemplo, ¿qué debería hacer el programa? ¿Qué clases debes definir? ¿Qué métodos debes implementar para estas clases?

En este último laboratorio el objetivo era implementar las funciones del 3er laboratorio en c++ quitando las distintas competiciones como groupPlay, League y cup. Para hacerlo tan sólo hemos tenido que traducir el código de las clases y generar 2 archivos por cada clase hija, osea outfielder y goalkeeper. .hpp son archivos de cabecera que contienen las declaraciones de clases, funciones y variables. Los archivos .cpp, por otro lado, son archivos de código fuente que contienen las definiciones o implementaciones de las clases, funciones y variables declaradas en los archivos .hpp
En cuanto a la traducción el reto era pasar la sintaxis típica de métodos a funciones y los diferentes if, else o for que han tenido que ser modificados. Aún así la parte que más problemas nos ha causado ha sido el cambio a punteros para acceder a los objetos.

Las clases son las siguientes: country, match, team y player que es el padre de outfielder y goalkeeper.

La clase Player junto a sus hijos **Outfilder y Goalkeeper. Player** sigue teniendo sus getters junto a un update stats, cabe remarcar que solo tiene los atributos comunes de sus hijos.

Las clase **Outfilder y Goalkeeper** tienen el mismo constructor de player y cada uno tiene unos atributos específicos en el portero tenemos saves y goles encajados y en outfielder shoots y tackles entre otros. En cuánto a métodos tienen los getters de los atributos y un update stats específico con los atributos no comunes.

La clase **team** tiene un addPlayer() y los métodos getters a parte del constructor

Country: Esta clase sigue vacía con su único getter

Match: Tiene los métodos printMatch y simulateMatch a parte de su constructor y los getters.

Finalmente en el main creamos jugadores algunos goalkeepers y la mayoría outfilders, países, equipos y diferentes partidos, pasando directamente los atributos del constructor manualmente.

Tras inicializar los objetos llamamos a los métodos de los partidos para simularlos e imprimir las estadísticas para comprobar que todo funciona correctamente.

2. Descripción de soluciones alternativas posibles que se discutieron, y una descripción de la solución elegida y la razón para elegir esta solución en lugar de otras. También es una buena idea mencionar los conceptos teóricos relacionados con la programación orientada a objetos que se aplicaron como parte de la solución.

En la implementación del código tuvimos bastantes dudas a la hora de implementar los punteros ya que al estar acostumbrados a Java en el cambio tuvimos que fijarnos en:
1. **Override**: En las clases `Goalkeeper` y `Outfielder`, que son subclases de `Player`, se utiliza la palabra clave `override` para indicar que el método `updateStats` está sobrescribiendo el método de la clase base `Player`. Esto es necesario porque `updateStats` es un método virtual puro en `Player`, lo que significa que las subclases están obligadas a proporcionar su propia implementación.

2. **La clase abstracta**: La clase `Player` es una clase abstracta en este código. Esto se indica mediante la presencia del método virtual puro `updateStats`. Como resultado, no puedes crear una instancia de `Player` directamente, sólo puedes usarla como clase base para otras clases como `Goalkeeper` y `Outfielder`.

Al convertir tu código de Java a C++, tuvimos que tener en cuenta varias diferencias clave entre los dos lenguajes. Por ejemplo, en C++, las variables miembro privadas de una clase no pueden ser accedidas directamente por las clases derivadas, a diferencia de Java donde pueden ser accedidas si están en el mismo paquete.

3. Conclusión: Describe qué tan bien funcionó la solución en la práctica, es decir, ¿mostraron las pruebas que las clases se implementaron correctamente? También puedes mencionar cualquier dificultad durante la implementación, así como cualquier duda que hayas tenido.
En el main fuimos probando con impresiones como de partidos como de estadísticas de equipo o jugadores.

```
Real Madrid 6 - 0 Rayo Vallecano
-----------------------------------
Crónica del Partido:
Gol Real Madrid--> Marcelo
Gol Real Madrid--> Benzema
Gol Real Madrid--> Casemiro
Gol Real Madrid--> Sergio Ramos
Gol Real Madrid--> Vinicius Jr.
Gol Real Madrid--> Benzema
```

Durante el proceso de conversión del código de Java a C++ y su posterior compilación en Visual Studio Code, nos encontramos con varios errores de compilación.

Uno de los errores que encontramos fue relacionado con el uso de la palabra clave `override` en C++. En C++, `override` se utiliza para indicar que un método en una clase derivada está destinado a sobrescribir un método en la clase base. Sin embargo, si el método de la clase base no es virtual o si la firma del método no coincide exactamente con la del método de la clase derivada, el compilador emitirá un error. Tuvimos que revisar

cuidadosamente nuestro código para asegurarnos de que estábamos utilizando `override` correctamente.

Otro error que encontramos fue relacionado con los punteros. Tuvimos que revisar nuestro código para asegurarnos de que todos nuestros punteros estaban correctamente inicializados y que no estábamos intentando acceder a memoria que ya había sido liberada.
Durante el proceso de conversión del código de Java a C++, me encontré con varios errores de compilación. Uno de los errores más comunes fue el error de “no encontrar la clase”.
Una causa común de este error es tener problemas con las directivas #include. Si la definición de la clase está en otro archivo y no estoy incluyendo ese archivo correctamente en mi código, el compilador no podrá encontrar la definición de la clase. Para solucionar este problema, tuvimos que asegurar de que se estaba utilizando la directiva #include correctamente y de que la ruta al archivo era correcta.

Aún asi en cuánto a dificultad ha sido para nosotros seguramente la práctica más sencilla ya que no hemos tenido que implementar ninguna clase ni utilidad nueva, ya estaba todo creado ha sido sólo cuestión de ir traduciendo a trozos e ir descubriendo las semejanzas.