LAB 1 POO; XAVI ALGARRA (U214401)
BIEL DOMINGO (u215110)

INFORME LABS 3 POO

1. 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?

El objetivo de la práctica era el de trabajar las herencias con clases padres y clases hijas. Concretamente 4 de las clases ya creadas anteriormente han sido clases padres y se han creado otras 6 hijas. Las 4 clases padres són Competición, Player, Team y Match; y las hijas son NationalTeam, Outfilder, Goalkeeper, Liga, Cup, GroupPlay y CupMatch.

La primera clase creada ha sido **competition**, cuyos atributos són name, country, gender, teams, matches y clubs. El constructor consta del nombre, country, clubs y género por inicializar. En cuanto a atributos tenemos los getters (ya vistos en todas las otras clases).
→ **addTeam()**: que comprueba el género del equipo y si se trata de selección nacioanl para que se corresponda con la liga.
→ **generateMatches():** que genera todos los partidos de la liga ida y vuelta.
→ **printRounds():** que imprimiría los partidos a jugar en la liga.
→ **simulateMatches():** llama a la simulación de partidos generados por la liga dando resultado aleatorio, goleadores y pases de goles simulados de la misma manera.
→ **printMatches():** imprime la suma de partidos de la liga con resultado y goleadores.
→ simulateSingleMatches que simula solo un partido no toda la liga y printSingleMatches.

Las 3 clases hijas de estas han sido **Cup, GroupPlay** i **League.**

La clase **League** se queda igual que en la otra práctica con la única diferencia que los atributos y métodos comunes de las 3 competitions se quedan en la clase padre y league los hereda directamente por lo tanto los hemos scado

La clase **GroupPlay:** Representación de una competición de grupos en un deporte, donde los equipos compiten en diferentes grupos antes de avanzar a rondas posteriores.

1. Constructor `GroupPlay`:
- Crea una instancia de la clase "GroupPlay" con un nombre, país, género y un indicador de si los equipos son clubes o selecciones nacionales.
- Inicializa el número de grupos en 2 y la lista de grupos (`groups`).

2. Método `generateMatches`:
- Crea grupos y distribuye equipos aleatoriamente en esos grupos.
- Los equipos se mezclan aleatoriamente y se distribuyen en grupos (`Leagues`).

3. Método `printGroups`:
- Muestra por pantalla los grupos en la competición.
- Imprime el nombre de la competición y los equipos en cada grupo.

4. Método `printGroupPlayMatches`:
- Imprime los partidos de grupo en la competición.

- Itera a través de los grupos y muestra los partidos que se han generado.

5. Método `simulateMatches`:
- Simula los partidos en cada grupo de la competición.
- Itera a través de los grupos y llama al método `generateMatches` y `simulateMatches` en cada grupo para simular los partidos de grupo.

La clase **Cup**: Representa una competición de copa en la que los equipos compiten en rondas sucesivas hasta que se determina un ganador. El código incluye la lógica para generar y simular los partidos de la copa, así como para determinar al ganador al final de la competición.

La clase Cup representa una competición de eliminación directa en la que los equipos compiten en varias rondas hasta que solo queda un ganador.

Aquí te explico cada parte del código:

Constructor Cup: Crea una instancia de la clase Cup con un nombre, país, género y un indicador de si los equipos son clubes o selecciones nacionales. Inicializa las matrices de equipos (tr) y partidos de copa (mr).

Método TeamstoCompetition: Agrega todos los equipos existentes a la primera ronda de la competición.

Método generateMatches: Genera y simula los partidos de cada ronda. Si el número de equipos es impar, el último equipo pasa automáticamente a la siguiente ronda. Los equipos se mezclan aleatoriamente y se emparejan para competir en partidos. Los partidos se simulan y los ganadores avanzan a la siguiente ronda.

Método simulateMatches: Imprime el nombre del equipo ganador de la competición. Si no se han simulado partidos, muestra un mensaje indicando que no se han jugado partidos en la competición.

Otra clase que ha cambiado es Player junto a las nuevas **Outfilder y Goalkeeper.** La clase player le ha pasado lo mismo que a league, se ha quedado prácticamente igual el único cambio ha sido el de quitar los métodos y atributos no comunes entre las dos clases hijas como por ejemplo tackles o pases que solo están en outfilder.

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.

Las clases **team** y **nationalTeam** són idénticas entre ellas solo cambiamos el addPlayer() en el de nationalTeam añadimos otro if mirando si la nacionalidad coincide con e¡la selección cosa que en team no es necesario. El team se queda igual que en el Lab2.

Country i **Match** se quedan igual solo añadimos una clase hija a **Match** que es **CupMatch.** Esta tiene mismo constructor y cambiamos el método simulate match para que no pueda haber empate, si lo hay se disputa la prórroga.

Finalmente en el main creamos jugadores algunos goalkeepers y la mayoría outfilders, países, equipos y selecciones nacionales y diferentes competiciones como liga, cup y match cup, pasando directamente los atributos del constructor manualmente.

Tras inicializar los objetos llamamos a los métodos de las competiciones para crear los partidos 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 el diseño presentado en la explicación de este Laboratorio daba una guia de qué atributos tiene que tener cada clase. En gran parte hemos seguido la guía pero en alguna clase hemos debatido en si añadir o quitar para poder completar toda la implementación.
Uno de los casos ha sido en el CupMatch.java, explicada anteriormente. En esta, hemos decidido añadir los atributos booleanos de prórroga y penalties, utilizados en la simulación de partidos de copa para poder almacenar si el encuentro llegó a la prórroga y si tuvo penales. Esto nos ha ayudado a la hora de la impresión de partidos.

Durante la elaboración del código, otro punto a debate fue la manera de implementar los métodos que están en padre e hijos con diferentes funciones. Finalmente decidimos usar el @Override, esto en Java se usa para decirle al programa que estás reemplazando un método de una clase superior con un método en una clase inferior. Esto asegura que estés haciendo la sustitución correctamente y ayuda a prevenir errores. Una de las funciones donde usamos esta herramienta fue en updateStats() implementado en la clase Player.java pero usado con @Override en sus clases hijas, Outfielder.java y Goalkepeer.java con diferentes usos en cada dedicado a las estadísticas personalizadas de cada subclase.

Otra clase en la que tuvimos debate de cómo implementarla fue la de Cup.java. Esto llega en el planteamiento de como colocar los partidos de los equipos participantes para poder eliminar los perdedores y continuar hasta llegar a un ganador. Pensamos en añadir un nuevo atributo para poner la ronda en la que está cada equipo, hacer una array de equipos ganadores y perdedores, intentar recursividad. Finalmente llegamos a una solución más sencilla, tener una lista auxiliar para poder añadir los equipos ganadores más un posible

último de la lista en el caso del número de equipos sea impar. Después cambiarla a la lista del atributo de Cup y continuar en el bucle hasta que la lista de ganadores sea solamente 1. Igualmente, esta no sería nuestra decisión final sobre esta parte del código, releyendo las instrucciones de la práctica, vimos que comentaba registrar dos matrices bidimensionales, una para los partidos y otra para los equipos. De esa manera se registran todos los partidos de la copa y no hay que tener una lista auxiliar o ir eliminando la lista de partidos como hacíamos anteriormente.

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 de las competiciones para comprobar que toda la implementación estuviera correctamente. Impresiones de partidos como esta:

```
Partido de Copa del rey que enfrentó a:

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
El partido no tuvo que llegar a la prorroga.
```

La implementación tuvo algunos errores en los primeros intentos, la mayoría por la poca familiarización con la herencia. Algún constructor nos salía en rojo y eso es porque en el super donde buscábamos los atributos de la clase padre habían atributos en diferentes posiciones del constructor.

Las clases que más nos ha costado implementar han sido Group Play y Cup. En el caso de group play el error que teníamos era que el número de grupos empezaba en 0 y cuando dividimos el total de equipos por los grupos nos daba error al dividir por 0, finalmente iniciamos el número de grupos manualmente. En cup la mayor dificultad era la de implementar la matriz que contenga los equipos que pasan de ronda. Finalmente no obviamos esa posibilidad y conseguimos mostrar en consola cada ronda de la competición.

Las demás clases las hemos implementado sin mayores inconvenientes, usando muchas pruebas y outputs en el main para irnos orientando de los métodos que funcionan perfectamente y los que se pueden mejorar.