LAB 4 - PROGRAMACIÓN ORIENTADA A OBJETOS
Implementación de interfaces

El objetivo de este laboratorio es implementar un mecanismo para la clasificación de la tablas y la lista de goleadores de league. Para hacerlo posible, se ha de desarrollar la implementación del diseño previamente hecho en el seminarios 4, rescatando el código hecho en la práctica 3 y realizando una série de cambios. Este consta de implementar las clases TeamStats, GoalkeeperStats, OutfielderStats, la clase abstracta, PlayerStats y la interfaz Comparable.

Ha sido necesaria la redefinición de los atributos de algunas clases como Team, Player, Goalkeeper y Outfielder.
La información de relación entre clases facilitada por el profesorado a parte de la ya definida en la práctica anterior establece que: Las clases GoalkeeperStats y OutfielderStats tienen una relación de herencia con la clase PlayerStats. Por otro lado, las clases TeamStats y PlayerStats tienen una relación de realización/implementación con la clase Comparable.

Para empezar con la implementación del código, empezamos con la implementación de la clasificación de las tablas de league. Esta clasificación sigue un criterio en la que los equipos son clasificados según:

1. Más puntos, definidos como $3 * \text{noWins} + \text{noTies}$.
2. Mejor diferencia de goles, definidos como goalsFor − goalsAgainst.
3. Más goles marcados, es decir, goalsFor.

```java
@Override
public int compareTo(TeamStats other){

    //miramos que equipo tiene más puntos con la formula 3*noWins +noTies
    int pointsTeam = 3*noWins +noTies;
    int pointsOther = 3*other.noWins + other.noTies;

    if(pointsTeam < pointsOther) return 1;     //menos puntos
    else if (pointsTeam > pointsOther)return -1;  //mas puntos
    else {     //mismos puntos

        //miramos que equipo tiene la better goal difference
        int goalDifferenceTeam = goalsScored - goalsAgainst;
        int goalDifferenceOther = other.goalsScored - other.goalsAgainst;

        if(goalDifferenceTeam < goalDifferenceOther) return 1;     //diferencia menor
        else if (goalDifferenceTeam > goalDifferenceOther)return -1;  //diferencia mayor
        else { //misma diferencia

            //miramos los goles metidos
            if(goalsScored < other.goalsScored) return 1;     //diferencia menor
            else if (goalsScored > other.goalsScored)return -1;  //diferencia mayor
            else return 0;   //el único caso en el que devolverá 0, ya que en los tres escenarios las s

        }
    }
}
```

(en caso de que dos equipos no se diferencien en estás tres categorías, se clasifican arbitrariamente)

Por tanto en la clase TeamStats se hace un @Override de la función compareTo (de la clase Comparable), para clasificar los equipos correctamente.

Para implementar la clase Comparable hay dos formas en java para implementar esta interfaz. Nosotras hemos optado por la implementación de Comparable<TeamStats> y @Override compareTo. En este caso no ha sido necesario realizar un downcast a TeamStats antes de realizar la comparación.

En el caso de la clasificación de goleadores, se ha de comparar el atributo noGoals de cada outfielder que se encuentra en la clase OutfielderStats. Esto es posible, con un @Override del método compareTo en la clase OutfielderStats.

```
@Override
public int compareTo(OutfielderStats other) {

    //miramos quien a maracado más goles
    if(noGoals < other.noGoals) return 1;     //menos goles
    else if (noGoals > other.noGoals)return -1;  //mas goles
    else return 0;    //mismos goles
}
```

El siguiente paso, como hemos mencionado, ha sido precisa la modificación de atributos en las clases Team y Player. Ha sido precisa la definición y por tanto la implementación en los constructores de Team y Player, de un diccionario desde Competition hasta Team/PlayerStats.

Para la implementación de los diccionarios, hemos usado la clase existente en Java, HashMap.

```
protected HashMap< Competition, PlayerStats> statsPlayer;
```

El método update de Team y Player, actualiza las estadísticas del equipo y jugador respectivamente. Esto se hace mirando primero en el diccionario para ver si el equipo o jugador tiene alguna estadística asociada con la competition. En caso que no la tenga, hemos creado una nueva instancia en la clase Team/PlayerStats y la hemos añadido al diccionario para la competition que se está jugando.

En el caso de la clase Team, luego simplemente llamamos al método updateStats de TeamStats con el match dado. Como anteriormente, update también llama al método de update de Player para todos los jugadores del equipo.

En cambio, en el caso de la clase Player, las clases Goalkeeper y Outfielder hacen un @Override del método update. La razón es que el método puede crear una instancia de GoalkeeperStats si el jugador es Goalkeeper y una instancia de OutfielderStats si el jugador es Outfielder.

```java
@Override
public void updateStatsPlayer(Match m, Competition c){
    //miramos si hay estadisticas del jugador asociadas a la competición
    if (statsPlayer.containsKey(c) == false){

        // y añadimos los nuevos stats asociados a la competición en el diccionario
        GoalkeeperStats playerSt = new GoalkeeperStats(this);
        statsPlayer.put(c, playerSt);
    }

    //actualizamos las estadisticas
    statsPlayer.get(c).updateStats(m);
}
```

```java
@Override
public void updateStatsPlayer( Match m, Competition c){

    //miramos si hay estadisticas del jugador asociadas a la competición
    if (statsPlayer.containsKey(c) == false){

        // y añadimos los nuevos stats asociados a la competición en el diccionario
        OutfielderStats playerSt = new OutfielderStats(this);
        statsPlayer.put(c, playerSt);
    }

    //actualizamos las estadisticas
    statsPlayer.get(c).updateStats(m);
}
```

Cuando las estadísticas del jugador existen en el diccionario, luego simplemente llamamos al método updateStats de PlayerStats con el match dado. Como anteriormente, la implementación de updateStats es distinta para Goalkeeper (implementada en GoalkeeperStats) y Outfielder (implementada en OutfielderStats).

Por último, para mostrar por pantalla la tabla de league y la lista de los goleadores, hemos empezado con la implementacion del método printTable en la clase League, creando una lista de TeamStats conteniendo las estadísticas de cada equipo de la Competition actual. Para ello hemos ordenado la lista de TeamStats utilizando el método Collections.sort. Para el formato de la tabla, simplemente le pedimos a ChatGpt que nos proporcionara un formato para imprimir los resultados de la liga, con ese formato lo adaptamos a nuestras necesidades.

```java
//imprimimos stats
System.out.println();
System.out.println(" TABLE LEAGUE: " + getName() + "\n");
System.out.printf(format:"%-25s | %-8s | %-4s | %-4s | %-4s | %-13s | %-15s%n",
    ...args:"Team", "Matches", "Wins", "Ties", "Losses", "Goals Scored", "Goals Against" + "\n");
```

```
FINAL STATS THE LEAGUE:

 TABLE LEAGUE: La Liga
Team                    | Matches  | Wins | Ties | Losses | Goals Scored  | Goals Against
FC Barcelona            | 4        | 2    | 1    | 1      | 9             | 11
Real Madrid             | 4        | 1    | 2    | 1      | 13            | 12
FC Barcelona Femenino   | 4        | 1    | 1    | 2      | 10            | 9
```

Finalmente, hemos implementado el método printGoalScorers de la clase Competition. Para hacerlo, hemos implementado un método getOutfielderStats, que devuelve una lista de OutfielderStats para todos los jugadores Outfielder de todos los equipos de la competition. Una vez que tenemos dicha lista, simplemente la ordenamos usando Collections.sort e imprimimos los primeros k jugadores (usando el método printStats de OutfielderStats).

Para dar por finalizada la práctica, no ha sido necesario añadir gran cosa en nuestra clase de prueba TestApplication. Hemos cambiado la forma de acceder a las estadísticas de los equipos en cada tipo de competición. Para ello, corregimos el hecho de tener que pasar por el diccionario para poder acceder a estos datos.
Durante el proceso, hemos ido compilando frecuentemente para la comprobación de errores.