LAB 3 - PROGRAMACIÓN ORIENTADA A OBJETOS
Implementación de herencia y polimorfismo

El objetivo de este laboratorio es seguir desarrollando la implementación del diseño previamente hecho en los seminarios 1, 2 y 3, rescatando el código hecho en la práctica 2 y realizando una série de cambios. Este consta de implementar las clases NationalTeam, CupMatch, Goalkeeper, Outfielder, Competition, Cup, GroupPlay y la relación de herencia que la mayoría de ellas tienen con clases creadas en prácticas anteriores.

Aún y aprovechando el código implementado en la práctica 2, ha sido necesaria la redefinición de algunas clases como League y Team.
La información de relación entre clases facilitada por el profesorado a parte de la ya definida en la práctica anterior define que: La clase NationalTeam tiene una relación de herencia con la clase Team y CupMatch tiene una relación de herencia con la clase Match. Por otro lado, Goalkeeper y Outfielder tienen una relación de herencia con la clase Player. Por último, League, Cup y GroupPlay tienen una relación de herencia con la clase Competition.

Sabiendo esto, la clase NationalTeam no contiene atributos adicionales a los que hereda de Team pero como adición tiene los métodos: *NationalTeam()* su constructor que cogerá como atributos *name*, *country* y *gender*. Por otro lado el método *addPlayer()* que coge como atributo player y tiene este método debido a que no puede heredarlo de la clase Team ya que la condición para poder añadir un jugador al equipo es que sea de la misma nacionalidad cosa que en *addPlayer()* de Team no es condición necesaria.

```
public void addPlayer(Player p){
    if (p.getNationality() == country){
        if (p.isFemale() == false && gender == Gender.HOMBRE) {
            players.add(counterPlayer,p);
            counterPlayer++;
            System.out.println(p.getName() + " added sucessfully");

        } else if(p.isFemale() == true && gender == Gender.MUJER){
            players.add(counterPlayer,p);
            counterPlayer++;
            System.out.println(p.getName() + " added sucessfully");

        }
        else if((p.isFemale() == true || p.isFemale() == false) && gender == Gender.MIXTO){
            players.add(counterPlayer,p);
            counterPlayer++;
            System.out.println(p.getName() + " added sucessfully");

        } else {
            System.out.println(p.getName() + " can not be added due to their gender.");
        }

    } else {
        System.out.println(p.getName() + "can not be added due to their nationality.\n");
    }

}
```

(NationalTeam)

```java
//methods
public void addPlayer(Player p){
    if (p.isFemale() == false && gender == Gender.HOMBRE) {
        players.add(counterPlayer,p);
        counterPlayer++;
        System.out.println(p.getName() + " added sucessfully");


    } else if(p.isFemale() == true && gender == Gender.MUJER){
        players.add(counterPlayer,p);
        counterPlayer++;
        System.out.println(p.getName() + " added sucessfully");

    }
    else if((p.isFemale() == true || p.isFemale() == false) && gender == Gender.MIXTO){
        players.add(counterPlayer,p);
        counterPlayer++;
        System.out.println(p.getName() + " added sucessfully");

    } else {
        System.out.println(p.getName() + " can not be added due to their gender.");
    }
}
```

(Team)

Como podemos observar nosotras para verificar que el jugador que quiere añadirse a NationalTeam tiene la misma nacionalidad, hemos optado por la comparación de nombres de los *Country* del *Team* y del *Player* en vez de reescribir la función *equals* que *Country* hereda de *Object,* como sugiere el profesorado.

Además, con la creación de la subclase NationalTeam, se tuvo que añadir el atributo booleano clubs a Competition, este servirá para saber si la competición es del tipo club competition o international competition, con la adición de este atributo hemos modificado la función *addTeam* de para que tenga en cuenta el parámetro de clubs a la hora de añadir un equipo a una determinada competición.

```java
public void addTeam(Team t){
    //si clubs es international competitions y team es un nationalTeam(jugadores son del mismo pais)
    if(clubs == true && t instanceof NationalTeam){
        if (t.getGender() == Gender.HOMBRE && gender == Gender.HOMBRE) {
            teams.add(counterTeams, t);
            counterTeams++;
            System.out.println(t.getName() + " added sucessfully");


        } else if(t.getGender() == Gender.MUJER && gender == Gender.MUJER){
            teams.add(counterTeams, t);
            counterTeams++;
            System.out.println(t.getName() + " added sucessfully");

        }
        else if((t.getGender() == Gender.HOMBRE || t.getGender() == Gender.MUJER) && gender == Gender.MIXTO){
            teams.add(counterTeams, t);
            counterTeams++;
            System.out.println(t.getName() + " added sucessfully");

        } else {
            System.out.println(t.getName() + " can not be added due to their gender type.");
        }
    }

    //si clubs es club competitions y team no es nationalTeam(nos da igual de que país son los jugadores))
    else if(clubs == false && !(t instanceof NationalTeam)){
        if (t.getGender() == Gender.HOMBRE && gender == Gender.HOMBRE) {
            teams.add(counterTeams, t);
            counterTeams++;
            System.out.println(t.getName() + " added sucessfully");


        } else if(t.getGender() == Gender.MUJER && gender == Gender.MUJER){
            teams.add(counterTeams, t);
            counterTeams++;
            System.out.println(t.getName() + " added sucessfully");

        }
        else if((t.getGender() == Gender.HOMBRE || t.getGender() == Gender.MUJER) && gender == Gender.MIXTO){
            teams.add(counterTeams, t);
            counterTeams++;
            System.out.println(t.getName() + " added sucessfully");

        } else {
            System.out.println(t.getName() + " can not be added due to their gender type.");
        }
    }
    else{
        System.out.println(t.getName() + " can not be added due to the type of competition.\n");
    }

}
```

Por otro lado, la clase CupMatch no contiene atributos adicionales a los que hereda de Match pero como adición a parte de su constructor *CupMatch()* que coge por parámetros *homeTeam* y *awayTeam*, esta clase tiene el método: *simulateMatch()* con la distinción que los equipos no pueden acabar un partido con un empate. A parte, también se indica por pantalla cuando dos equipos quedan empate y los goles son nuevamente asignados aleatoriamente, ya que en una Cup dos equipos no pueden acabar un partido en empate.

```
Round 2:
Match 1
Home goals: 0      Away goals: 0
Tie, extra time begins:
Home goals: 0      Away goals: 0
Tie, penalties:
Home goals: 5      Away goals: 4
```

Al simular partidos tanto de tipo cupMatch y Match, nos dimos cuenta de que al asignar goles aleatoriamente se tenían en cuenta los jugadores del tipo *GoalKeeper*, para solucionar esto, ha sido preciso crear dos listas, *homeOutfielders* i *awayOutfielder*, la cual guarda los ganadores de cada equipo para distribuir aleatoriamente los goles únicamente a esos jugadores que pertenecen a la clase Outfielder. Esto se ha implementado en el método de siumulateMatch() en la clase Match i CupMatch.

```java
//Cogemos los outfilders de los teams para asiganr los goles
LinkedList<Player> homeOutfilders = new LinkedList<Player>();
LinkedList<Player> awayOutfilders = new LinkedList<Player>();

//HomeTeam:
for(int i = 0; i < homeTeam.getSizeListPlayers(); i++){
    Player p = homeTeam.getPlayer(i);
    int index = 0;
    if(p  instanceof Outfielder){
        homeOutfilders.add(index, p);
        index++;
    }
}
//AwayTeam:
for(int i = 0; i < awayTeam .getSizeListPlayers(); i++){
    Player p = awayTeam.getPlayer(i);
    int index = 0;
    if(p  instanceof Outfielder){
        awayOutfilders.add(index, p);
        index++;
    }
}

//Distribuimos de manera aleatoria los goles en los jugadores outfilder
for(int i = 0; i < homeGoals; i++){
    int randomindex = random.nextInt(homeOutfilders.size());
    Player p = homeOutfilders.get(randomindex);

    homeScorers.add(i, p.getName());
}

for(int i = 0; i < awayGoals; i++){
    int randomindex = random.nextInt(awayOutfilders.size());
    Player p = awayOutfilders.get(randomindex);

    awayScorers.add(i, p.getName());
}
```

Con respecto a la clase Player, esta tiene dos subclases: Goalkeeper y Outfielder.
La clase Goalkeeper a parte de heredar los atributos de su clase padre, tiene dos atributos adicionales *noSaves* i *noGoalsLet.* Esta clase tiene su constructor: *Goalkeeper()* que coge como parámetros *gender*, *name, age* i *nationality*; y como método adicional *updateStats()* al que le entra como parámetro un *match*.

Paralelamente, la clase Outfielder tiene atributos adicionales *noTackles, noPasses, noShots, noAssists* i *noGoals.* Esta clase tiene su constructor: *Outfielder()* que coge como parámetros *gender*, *name, age* i *nationality*; y como método adicional *updateStats()* al que le entra como parámetro un *match*.
Estas dos clases hijas de Player tienen un método adicional de *updateStats()* ya que cada clase tiene sus propios parámetros que actualizar.

Por último, la clase Competition ha sido implementada aprovechando código de la práctica 2 para la clase League. Se han aprovechado los atributos *clubs, name, country, gender, teams* y *matches* pero tiene como adición el atributo *clubs* de tipo *bool*. Esta clase a parte de su constructor *Competition()* al que le entran por parámetros *club, name, country* i *gender;* tiene por métodos los getters, *addTeam* que le entra por parámetro un Team, *generateMatches()*, *simulateMatches(), printMatches* i *printGoalScorers* al que le entra por parámetro un entero.

Las clases Cup, League i GroupPlay heredan los atributos y la mayoría de métodos descritos anteriormente en la clase Competition.
La Clase Cup tiene como atributos adicionales a los ya contenidos en Competition *tr (una lista conteniendo los Team participando en ese momento)* i *mr (una lista conteniendo los Match realizados).* Como constructor *Cup()* recibe como parámetros *name, country* i *gender*. Tiene como métodos adicionales *generateMatches()* i *simulateMatches()* que aún y estando implementados en la clase Competition, esta clase tiene una condición para estas funciones distinta al método implementado en Competiton. A parte también tiene adicionalmente el método *printBracket().*
La Clase League no tiene atributos adicionales a los que puede heredar de la clase Competition. Como constructor *League()* recibe como parámetros *name, country* i *gender*. Tiene como método adicional *generateMatches()* ya que tienen una condición distinta al método implementado en Competition y por otro lado el método *printTable()*.
La Clase GroupPlay tiene como atributos adicionales a los ya contenidos en Competition *noGroups* i *groups (una lista del tipo League).* Como constructor *GroupPlay()* recibe como parámetros *name, country* i *gender*. Tiene como método adicional *generateMatches()* i *simulateMatches()* que aún y estando implementados en la clase Competition, esta clase tiene una condición para estas funciones distinta al método implementado en Competition. A parte también tiene adicionalmente el método *printTables()* al igual que la clase League y por eso comparten una relación de agregación.

Al finalizar la implementación, hemos definido la clase de prueba TestApplication. Con un método principal, hemos creado instancias de las diferentes clases, aplicando los diversos métodos. Hemos creado instancias de country, jugadores, equipos (teniendo en cuenta national teams) y ligas.Para Cup hemos creado una copa, añadido tres equipos a esta (hay otro comentado en el código), simulado y generado los partidos para después imprimir los estados de cada jugador y equipo, se han escogido tres equipos para que se vea el caso de tener equipos impares. Los mismos pasos hemos seguido para League. Para GroupPlay hemos creado dos ligas, una con tres equipos y otra con dos, para así poder comprobar que funcionan independientemente y ejecutan el número de partidos correctamente, para después poder seguir la misma estrategia implementada en Cup i League. Durante el proceso, hemos ido compilando frecuentemente para la comprobación de errores.