Martí Esquius Arnau, u213923, marti.esquius01@estudiant.upf.edu
Marc Guiu Armengol, u215107, marc.guiu01@estudiant.upf.edu

Lab5 Documentation

El objetivo de este laboratorio es implementar una aplicación de fútbol en C++, centrándonos en las clases Match, Country, Team, Player, Goalkeeper y Outfielder. Se proporciona el código base para Match, Country y Team, y se nos pide implementar las clases restantes junto con una función principal para realizar pruebas.

La clase **Match** cuenta con los siguientes atributos:

- **teamOne:** Team -> Uno de los equipos que se disputan el partido, el local.
- **teamTwo:** Team -> El otro equipo, visitante.
- **goalOne:** int -> Goles hechos por el equipo local.
- **goalTwo:** int -> Goles hechos por el equipo visitante
- **scorersOne:** list of Player -> Linked List que almacena los goleadores (jugadores que marcan goles) del equipo local.
- **scorersTwo:** list of Player -> Linked List que almacena los goleadores (jugadores que marcan goles) del equipo visitante.

Y los métodos:

- **Match**(h:Team, a:Team) -> Método constructor de la clase, recibe 2 equipos a los que trata como local y visitante. Asigna a homeGoals y awayGoals un valor de 0, e inicializa las dos LinkedList que almacenarán las listas de los jugadores goleadores respectivamente.
- **simulateMatch**() -> Este método genera un partido con un numero aleatorio de goles entre 0 y 6 para cada equipo. Una vez obtenidos los goles copia de forma aleatoria tantos jugadores como sea necesario ‘x’ desde el equipo que ha marcado ‘x’ goles (clase que contiene una Linked List de jugadores) hasta la linked list de goleadores dentro del Match.
  Finalmente hace un updateStats de cada equipo pasando el partido como atributo.
- **printMatch**() -> Este método imprime todo lo relacionado con un partido, los equipos que están jugando, la puntuación final y los goleadores de cada equipo.

Luego está la clase **Team**, con los atributos:

- **name**: String -> Nos dice el nombre del equipo en un String
- **country**: Country -> Nos dice el país del equipo en una clase Country.

- **gender**: Gender-> Nos dice el género de los jugadores (MALE, FEMALE o MIXT), en una enumeración que hemos definido previamente en el archivo Gender.java.
- **players**: list of Player -> Es una lista de los jugadores del equipo dónde cada uno de ellos es una clase Player.

Con siguientes métodos:

- **Team**(n:String, c:Country, g:Gender) -> Este es el método constructor, que inicializa todos los atributos del equipo, donde n es un String que se asocia al nombre (name), c es una clase Country que se asocia al país (Country) y g es una de las posibilidades de la enumeración Género que se asigna al género del equipo (gender). Los otros atributos se inicializan en 0 ya que se supone que el equipo no ha jugado ningún partido todavía. También inicializa la lista de jugadores.
- **getName**(): String -> Nos devuelve un String con el nombre del equipo.
- **getCountry**(): Country -> Nos devuelve una clase Country con el país del equipo.
- **getGender**(): Gender -> Nos devuelve el género del equipo (una de las tres posibilidades de la enumeración).
- **addPlayer**(Player p): boolean -> Si se cumplen las condiciones (mismo género del equipo y del jugador, o por otro lado que el equipo sea mixto), añade al jugador en el equipo y devuelve TRUE para verificar que se ha podido hacer, en caso contrario simplemente devuelve FALSE.

También la clase **Country**, con el atributo:

- **name**: String -> El nombre del país.

Y los métodos:

- **Country**(n:string) -> Método constructor, asigna el nombre del país.
- **getName**(): String -> Método getter, devuelve el nombre del país.
- **equals**(o:Object): Bool -> Compara dos países, para saber si son o no el mismo país.

Luego, nosotros hemos tenido que implementar las siguientes clases:

La clase abstracta **Player** con los atributos:

- **female**: bool -> Nos dice el género del jugador (TRUE si es chica y FALSE si es chico)

- **name**: String -> Nos dice el nombre del jugador en un String.
- **age**: int -> Nos dice la edad del jugador.
- **nationality**: Country -> Nos dice la nacionalidad del jugador, en la clase, que ya nos venía programada, Country.
- **noMatches**: int -> Nos dice el número de partidos que lleva el jugador en un int.

Y los métodos:

- **Player**(g:bool, n:String, a:int, nat:Country) -> Este es el método constructor, inicializa todas los atributos de la clase, asignando las variables recibidas a sus correspondientes atributos (female, name, age y nationality respectivamente), y los demás los inicializa a 0 (ya que se supone que todavía no ha jugado ningún partido).
- **isFemale**(): bool -> Este método, nos dice el género del jugador, diciéndonos si es o no chica (TRUE o FALSE).
- **getName**(): String -> Nos devuelve un String con el nombre del jugador.
- **getAge**(): int -> Nos devuelve un int con la edad del jugador.
- **getNationality**(): Country -> Nos devuelve una clase Country con la nacionalidad del jugador.
- abstract **updateStats**(m:Match) -> Recibe un partido y aumenta en uno los partidos jugados por el jugador (es un método que llamaremos al final de simular un partido).
- abstract **printStats**() -> Imprime las estadísticas del jugador.

En segundo lugar, tenemos la clase **Goalkeeper**, que es una subclase de jugador, que tiene los siguientes atributos:

- **noSaves**: int -> Este atributo sirve para guardar el número de paradas que ha hecho el portero.
- **noGoalsLet**: int -> En este se guarda el número de goles que este ha recibido.

Y los siguientes métodos:

- **Goalkeeper**(g:int ,n:string ,a:int ,nat:Country) -> Este es el método constructor, recibe el género, nombre, edad y nacionalidad del portero e inicializa a 0 los otros atributos.
- **updateStats**(m:Match) -> Este método sirve para actualizar las estadísticas del portero después de un partido, aumenta el número de partidos y calcula el de paradas.
- **printStats**() -> Imprime las estadísticas del jugador.

Y **Outfielder** que es otra subclase de jugador, con los siguientes atributos:

- **noTackles**: int -> Este atributo sirve para almacenar el número de entradas del jugador.
- **noPasses**: int -> Este atributo sirve para almacenar el número de pases del jugador.
- **noShots**: int -> Este atributo sirve para almacenar el número de disparos a portería del jugador.
- **noAssists**: int -> Este atributo sirve para almacenar el número de asistencias (pases de gol) del jugador.
- **noGoals**: int -> Este atributo sirve para almacenar el número de goles del jugador.

Y los métodos:

- **Outfielder**(g:int, n:string, a:int, nat:Country) -> Este es el método constructor, recibe el género, nombre, edad y nacionalidad del jugador e inicializa a 0 los otros atributos.
- **updateStats**(m:Match) -> Este método sirve para actualizar las estadísticas del jugador después de un partido, aumenta el número de partidos y el número de goles para sumarlo
- **printStats**() -> Imprime las estadísticas del jugador.

La forma en la que se podría mejorar el código es expandiendo las clases como en prácticas pasadas para crear una aplicación más completa. También añadiendo métodos de simulación que simplificaran el Main, dejándolo todo resumido en unas pocas líneas de código.

Una vez implementadas las dos clases, hemos hecho una prueba para comprobar que todo funcionaba correctamente, creando una serie de jugadores y de equipos para probar todos los métodos creados a lo largo de todo el código. El resultado ha sido positivo y todos los métodos han tenido el comportamiento esperado.