Laboratorio 2

1. Una introducción donde se describe el problema. Por ejemplo, ¿qué debería hacer el programa? ¿Qué clases tienes que definir? ¿Qué métodos debes implementar para estas clases?

El objetivo de este laboratorio es recopilar las estadísticas de los equipos y jugadores según sus diversas competiciones en las que han estado involucrados guardadas en diccionario. Así, se tiene la información del equipo en cada competición jugada.
La meta de este programa es implementar un sistema para ordenar tablas de ligas y listas de goleadores en el contexto de cada competición de juego.

1. **Clase TeamStats:**

La clase TeamStats implementa la interfaz Comparable<TeamStats>, lo que significa que las instancias de `TeamStats` pueden ser comparadas entre sí para determinar cuál es mayor, menor o igual.

- **Atributos**: `team`, `matchesPlayed`, `wins`, `ties`, `losses`, `goalsScored`, `goalsAgainst`, `points`. Estas variables almacenan la información del equipo, el número de partidos jugados, victorias, empates, derrotas, goles marcados, goles en contra y puntos, respectivamente. Estaban en Team, pero ahora formaran parte de TeamStats.
- **Constructor**: El constructor `TeamStats(Team t)` inicializa la variable `team` con el equipo proporcionado.
- **updateStats(Match m)**: Este método actualiza las estadísticas del equipo basándose en el resultado de un partido. Comprueba si el equipo es el equipo local o visitante en el partido y actualiza las estadísticas en consecuencia.
- **PrintStats()**: Este método imprime las estadísticas del equipo en la consola.
- **compareTo(TeamStats o)**: Este método se utiliza para comparar dos objetos `TeamStats`. La comparación se basa en los puntos, la diferencia de goles y los goles a favor. Si todos estos criterios son iguales, los equipos se pueden ordenar de manera arbitraria.

2. **Clase Team**

La clase Team representa un equipo de fútbol ahora se implementa el atributo ‘stats’ siendo un HashMap para cada competición con sus respectivas estadísticas del equipo ‘Team’ en ella.

- **Atributos**: name, country, gender, players, stats. Estas variables almacenan el nombre del equipo, el país del equipo, el género del equipo, una lista de jugadores en el equipo y las estadísticas del equipo en varias competiciones, respectivamente.
- **Constructor**: El constructor Team(String name, Country country, Gender gender) inicializa las variables de instancia con los valores proporcionados.
- **Método addPlayer(Player player):** Este método agrega un jugador al equipo si el género del jugador es compatible con el género del equipo.
- **Método removePlayer(Player player):** Este método elimina un jugador del equipo.
- **Método update(Competition c, Match m):** Este método actualiza las estadísticas del equipo y de cada jugador en el equipo después de un partido en una competición específica.

La relación entre las clases Team y TeamStats es que un objeto Team contiene un objeto TeamStats para cada competición en la que participa. Cuando se juega un partido, el método update de la clase Team actualiza las estadísticas del equipo y de cada jugador en el equipo utilizando el método updateStats de la clase TeamStats.

3. **Clase PlayerStats y sus subclases, OutfielderStats y GoalkeeperStats:**

Las clases PlayerStats, OutfielderStats y GoalkeeperStats se utilizan para manejar las estadísticas de los jugadores en un equipo de fútbol.

- PlayerStats es una clase abstracta que proporciona la estructura básica para las estadísticas de un jugador. Tiene un método updateStats(Match m) que se sobrescribe en las subclases para actualizar las estadísticas específicas del jugador. También tiene un método PrintStats() que es abstracto y debe ser implementado en las subclases.
- OutfielderStats es una subclase de PlayerStats que representa las estadísticas de un jugador de campo. Incluye estadísticas como goles, pases, asistencias y entradas. El método updateStats(Match m) se sobrescribe para actualizar estas estadísticas específicas. Además, implementa el método PrintStats() para imprimir sus estadísticas específicas y el método compareTo(PlayerStats o) de la interfaz Comparable, que permite comparar las estadísticas de los jugadores en función de los goles marcados.
- GoalkeeperStats es otra subclase de PlayerStats que representa las estadísticas de un portero. Incluye estadísticas como paradas y goles en contra. Al igual que en OutfielderStats, el método updateStats(Match m) se sobrescribe para actualizar estas estadísticas específicas. También implementa el método PrintStats() para imprimir sus estadísticas específicas.

Estas clases permiten manejar y actualizar las estadísticas de los jugadores en un equipo de fútbol, y proporcionan métodos para imprimir estas estadísticas y comparar jugadores en función de sus estadísticas.

4. **Clase Player y sus subclases, Outfielder y Goalkeeper:**

La clase Player representa a un jugador de fútbol.

Atributos: gender, name, age, nationality, stats, team. Estas variables almacenan el género del jugador, el nombre del jugador, la edad del jugador, la nacionalidad del jugador, las estadísticas del jugador en cada competición y el equipo al que pertenece el jugador, respectivamente.

- **Constructor**: El constructor Player(boolean gender, String name, int age, Country nationality) inicializa las variables de instancia con los valores proporcionados.
- **Método update(Competition c, Match m)**: Este método es un método vacío que se sobrescribe en las subclases para actualizar las estadísticas del jugador después de un partido en una competición específica.
- **Método equals(Object o)**: Este método se utiliza para comparar dos objetos Player. Si los nombres de los jugadores son iguales, entonces los jugadores son iguales.

Las clases Goalkeeper y Outfielder son subclases de la clase Player y representan a un portero y a un jugador de campo, respectivamente. Ambas clases sobrescriben el método update(Competition c, Match m) de la clase Player para actualizar las estadísticas del jugador utilizando las clases GoalkeeperStats y OutfielderStats, respectivamente, que son subclases de la clase PlayerStats.

Por lo tanto, estas clases permiten manejar y actualizar las estadísticas de los jugadores en un equipo de fútbol, y proporcionan métodos para comparar jugadores en función de sus estadísticas. La relación entre estas clases y las clases PlayerStats, OutfielderStats y GoalkeeperStats es que un objeto Player contiene un objeto PlayerStats para cada competición en la que participa, y este objeto PlayerStats se actualiza después de cada partido. Las clases OutfielderStats y GoalkeeperStats proporcionan implementaciones específicas de PlayerStats para jugadores de campo y porteros, respectivamente.

5. **Clase Competition y sus subclases, League y GroupPlay:**

Las clases Competition, League y GroupPlay representan diferentes tipos de competiciones de fútbol.

- Competition es la clase base que proporciona la estructura básica para una competición. Tiene métodos como addTeam(Team team) para agregar un equipo a la competición, generateMatches() para generar partidos entre los equipos, printRounds() para imprimir los partidos, y simulateMatches(), printMatches(), **printLeagueTable(), printTopScorers(int k)** que son métodos vacíos y se sobrescriben en las subclases. Estos dos últimos serán los que implementemos en este laboratorio.
- League es una subclase de Competition que representa una liga de fútbol. Sobrescribe los métodos generateMatches(), simulateMatches(), printLeagueTable(), y printTopScorers(int k) de la clase Competition para proporcionar implementaciones específicas para una liga.
    - Método printLeagueTable(): Este método imprime la tabla de la liga. Primero, ordena los equipos en función de su rendimiento en la competición utilizando el método compareTo() de la clase TeamStats. Luego, imprime la tabla de la liga, que muestra los equipos en orden de su rendimiento en la competición. Para cada equipo, imprime el nombre del equipo, los puntos, las victorias, los empates, las derrotas, los goles a favor, los goles en contra y la diferencia de goles.
    - Método printTopScorers(int k): Este método imprime los k máximos goleadores de la competición. Primero, crea una lista de todos los jugadores que han marcado goles en la competición. Luego, ordena la lista de jugadores en función de los goles marcados utilizando el método compareTo() de la clase OutfielderStats. Finalmente, imprime los k máximos goleadores de la competición, mostrando el nombre del jugador, el equipo, los goles y los goles por partido. Si hay menos goleadores que k, imprime todos los goleadores.

- GroupPlay es otra subclase de Competition que representa una competición de fútbol con formato de grupos. Al igual que League, sobrescribe los métodos generateMatches(), simulateMatches(), printLeagueTable(), y printTopScorers(int k) de la clase Competition para proporcionar implementaciones específicas para una competición de grupos.
    - Método printLeagueTable(): Este método imprime la tabla de la liga para cada grupo en la competición. Primero, ordena los equipos en cada grupo en función de su rendimiento en la competición utilizando el método compareTo() de la clase TeamStats. Luego, imprime la tabla de la liga, que muestra los equipos en orden de su rendimiento en la competición. Para cada equipo, imprime el nombre del equipo, los puntos, las victorias, los empates, las derrotas, los goles a favor, los goles en contra y la diferencia de goles.
    - Método printTopScorers(int k): Este método imprime los k máximos goleadores de la competición. Primero, crea una lista de todos los jugadores que han marcado goles en la competición. Luego, ordena la lista de jugadores en función de los goles marcados utilizando el método compareTo() de la clase OutfielderStats. Finalmente, imprime los k máximos goleadores de la competición, mostrando el nombre del jugador, el equipo, los goles y los goles por partido. Si hay menos goleadores que k, imprime todos los goleadores.

2. Una descripción de posibles soluciones alternativas que se discutieron, y una descripción de la solución elegida y la razón por la que se eligió 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.

La interfaz compareTo era un de los objetos trabajados en clase que había que implementar, nuestra primera intuición fue la de crear un nuevo documento como para las demás clases y allí poner la información de la interfaz. Después nos dimos cuenta que la interfaz la teníamos que introducirla en las clases necesarias con un “implement” . Luego con un override personalizamos la ordenación según la necesidad, en equipos por puntos, +- y goles a favor y en jugadores con goles.
En cuánto a los métodos de ordenación de league y group play nuestra idea inicial era la de crear los métodos en la clase Competition, la clase padre y que de esta forma la hereden las clases hijas. Finalmente vimos que en el GroupPlay la ordenación hacía una

clasificación sola de todos los grupos. Nuestra solución fue la de dejar el método sorting en Competition vacía y con override personalizar GroupPlay y League. La única diferencia entre estos dos métodos és que en GroupPlay tenemos un for inicial que recorre los grupos de esta froma ordenamos cada grupo por separado.

En la programación orientada a objetos, una clase abstracta es una clase que no puede ser instanciada y se utiliza principalmente como una clase base. Los métodos abstractos son métodos que se declaran en una clase abstracta sin implementación, y deben ser implementados en cualquier clase concreta que herede de la clase abstracta. En el código, PlayerStats es una clase abstracta con métodos abstractos como updateStats(Match m) y PrintStats().

Las interfaces, por otro lado, son una forma de lograr la abstracción total (todos los métodos de una interfaz son abstractos) y también permiten la herencia múltiple, ya que una clase puede implementar múltiples interfaces. En el código, Comparable<TeamStats> es una interfaz que TeamStats implementa, proporcionando una implementación para el método compareTo(TeamStats o).

Las clases OutfielderStats y GoalkeeperStats son subclases de PlayerStats y proporcionan implementaciones específicas de los métodos abstractos definidos en PlayerStats. Esto es un ejemplo de polimorfismo.

3. Una conclusión que describa qué tan bien funcionó la solución en la práctica, es decir, ¿mostraron las pruebas que las clases fueron implementadas correctamente? También puedes mencionar cualquier dificultad durante la implementación, así como cualquier duda que hayas tenido.

En esta práctica la mayoría de implementaciones tenían un output bastante claro por lo tanto hemos podido comprobar prácticamente todo el código. Formatear este output ha sido de las cosas más complicadas, había que imprimir muchas cosas y queríamos que se viera bien claro que eran. Por ejemplo, en el output de print table queríamos ver las estadísticas más importantes de los equipos en una tabla bien clara, las columnas rectas y las estadísticas todas con los valores correctos. Para hacerlo tuvimos que ejecutar el programa muchas veces y ir tocando los espacios y ordenando correctamente. En conclusión el output tenía todo lo necesario y el funcionamiento era correcto junto a todas la otras prácticas que nos han llevado hasta aquí con un programa con funcionamiento excelente y un output digno de las competiciones, jugadores y equipos que hemos implementado.

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

```
----------------Champions League--------------------


La clasificación de Group 1:
Team                Points      Wins        Ties        Losses      Goals For   Goals Against   Goal dif (+/-)
-------------------------------------------------------------------------------------------------------------
1.Real Sociedad     9           3           0           1           15          15              0

2.Girona FC         6           2           0           2           17          14              3

3.Atlético de Madrid 3          1           0           3           12          15              -3
```