# SISTEMA DE GESTIÓN DEPORTIVA



Practica de Programación Orientada a Objetos 2024-2025

Universidad Politécnica de Madrid

E.T.S. de Ingeniería en Sistemas Informáticos

Departamento de Sistemas Informáticos

Estudiantes: Víctor Palmier (victor.palmier@alumnos.upm.es),

Alejandro Rico (alejandro.ricog@alumnos.upm.es),

Tigrán Oganesyan (t.oganesyan@alumno.upm.es)

Grupo: IWSIM22-G1

Profesor: Joaquín Gayoso

### Match

- +match(Player player1,Player player2):Match
- <getter>getPlayer1():String
- <getter>getPlayer2():String
- + toString():String

## $\Psi$

## **Player**

# name: String # score: Double

- +player(String name):Player
- <getter> getName(): String
- <getter> getScore():Double
- <setter> setName(String): void
- <setter> setScore(Double):void
- + toString(): String

## **CLI**

- +print(String):void
- +scanner():scanner
- +isName():boolean



## Logic

- +start(int participantes):void
- +createPlayer(String name): void
- +removePlayer(String name): void
- -removeMatches(String name): void
- +showPlayers(): void
- +setScore(String name, int score): void
- +rankPlayers(): void
- +showMatchups(): void
- +clearMatchups(): void
- +matchPlayers(String name1, String name2): void
- +randomMatchup(): void
- -exists(name):boolean
- -searchPlayer(name):Player
- -menu():void

## dde UMLEntregas 2 y 3 cli -scanner Scanner - spelCommand() String - print():void teamRemove +apply String +toString String +getCommand:String +apply String +bString String +getCommand:String playerCreate +apply String +toString String +getCommand:String logaut +toString String +apply String +getCommand:String +apoly Syring +boString Syring -boString Syring login +apply String +toString string +getCommand:String +apply String +toString String +getCommand:String tournament list +toString String eapply String +soarch(String);Team +addTeam;Team);vold +addFuryerToTeam;Puryer,Team); -getTeams();Team -seleteTeam/Team);vold TournamentController +addictionnament(Controller +addictionnament(Controller) -addictionnament(Controller) -addictionnament(Controller) -addiction(Controller) -addiction(Controller) -addiction(Controller) -addictionnament(String) -addictionnament rankAux(Tournament).String showmatchmakes(Tournament).String +search(String):Tournament -getInslance():TournamentController -password:String tornament -tornament - register(tournament);void register(team);void matchmakingMan(tournament);String CAIRDONIS SCORED POINTS MATCH\_WON ASISTS\_POINTS PAST\_TOURNAMENTS GENERATED\_MONEY -name.String - isMember[Payer]Boolean - ipsCategory(category]Double +RemovaTournament[Tourname +addTournament[Tournament) category +gaiValue(): Double > +seiValue(): vald -value:double

#### 1. Modelo vista controlador

Hemos decidido implementar 4 controladores PublicController, TournamentController, PlayerController y TeamController, que manejan las interacciones entre la vista y el modelo, sustituyendo a la clase Logic, implementando también la totalidad de la lógica correspondiente a cada controlador.

El modelo tiene 3 clases principales, User, que es la clase que va a a tratar con la persona logeada, Member, que va a tratar con los miembros de un torneo y Tournament, que es el torneo.

- Los users (clase abstracta) van a poder ser administradores o jugadores a través de una herencia. Esto es debido a que ambos objetos son de la misma naturaleza y comparten gran parte de sus métodos y atributos.
- Los members (interfaz) van a poder ser equipos o jugadores, haciendo así que los torneos puedan ser disputados por ambos
  - o Cabe destacar que un equipo va a estar compuesto por 2 o más jugadores
  - Hemos decidido crear además una enumerado con los tipos de categoría por los cuales se puede regir un torneo.
  - Cada jugador tiene una lista de categorías a la que puede pedir su valor o insertarle el valor, mientras que el valor de una categoría en un equipo es la media geométrica de cada uno de los jugadores del equipo en dicha categoría

#### 2.Uso de POO

Vamos a analizar esta parte con los principios más importantes para comprobar si los cumplimos:

- **Encapsulación:** Cada clase está diseñada con atributos privados o protegidos, mientras que los métodos públicos proporcionan acceso controlado a estos atributos.
- **Herencia:** Como ya hemos mencionado en el ejemplo, utilizamos la herencia en user y member además de en Command para hacer uso del patrón
- Polimorfismo: El mejor ejemplo es la clase command donde permitimos que se implementen distintos métodos adaptándonos a las necesidades específicas del objeto, ya que cada comando es diferente, pero tiene el mismo método. También vemos esta implementación en member con el método getCategory en player y team.
- **Abstracción:** Clases como Member o User son interfaces que permiten definir "plantillas" que otras clases pueden implementar.

#### 3. Principios SOLID:

Podemos asegurar que nuestro código cumple los principios SOLID debido a la utilización de patrones de diseño

- Single Responsability: Cada clase tiene una responsabilidad única. Por ejemplo,
   PlayerController se encarga de gestionar jugadores, TournamentController gestiona torneos, TeamController solo se encarga de equipos.
- Open/Closed: El diseño puede implementar nuevos comandos en cualquier momento añadiendo funcionalidad. El código está cerrado a la modificación de las clases existentes.
- **Liskov Substitution Principle:** Las implementaciones de interfaces como Command garantizan que cualquier clase que implemente esta interfaz puede sustituirse sin romper el sistema.
- Interface Segregation Principle: Las interfaces están diseñadas para ser específicas, como Command, evitando obligar a las clases a implementar métodos que no usan.
- Dependency Inversion Principle: Las dependencias principales parecen invertidas al utilizar controladores para interactuar con los modelos en lugar de un acoplamiento directo.

#### 4. Patrones de Diseño:

#### Command:

 Se observa en las clases que implementan Command (e.g., PlayerAdd, TeamCreate). Este patrón encapsula solicitudes como objetos, permitiendo que se parametrice la funcionalidad y se gestione de manera más flexible.

#### Singleton:

 Implementado en los controladores para eliminar las asociaciones entre controladores y en los comandos, simplificando enormemente la gestión de recursos compartidos.