UNIVERSIDAD ARGENTINA DE LA EMPRESA
FACULTAD DE INGENIERÍA Y CIENCIAS EXACTAS

TRABAJO PRÁCTICO

Paradigma orientada a objetos

Integrantes del Grupo:

Borra Agustín	- LU 1184249
Gaspar Lautaro Conde	- LU 1195534
Luciano Joaquin Tobias	- LU 1190555
Pelaccini Franco	- LU 1188236

Profesor: Coedo Maximiliano Gonzalo

Cuatrimestre: 2 – Año: 2025

Lógica de negocio:

El sistema permite administrar y llevar historial de jugadores, equipos, árbitros, sedes (con sus canchas y ocupación por día y horario) y competiciones (torneos y partidos sueltos).

1. Gestión de Participantes

El sistema no permite crear partidos de la nada, primero se deben crear los jugadores y equipos y tienen que estar guardados en la base de datos.

- **Jugadores y Árbitros:** Existen servicios específicos (JugadorService, ArbitroService) para registrar, modificar, eliminar y buscar personas.
- **Equipos:** La clase EquipoService hace que el sistema agrupe jugadores en equipos de a 2.

2. Gestión de la Competición

manejada principalmente por CompeticionService. El sistema permite dos formas de juego:

- **Torneos:** Competiciones que tienen una lista de partidos y equipos inscritos.
- **Partidos Sueltos:** Da la capacidad de registrar un juego amistoso sin que pertenezca a un torneo (registrarPartidoSuelto).

3. Gestión de Espacios

El negocio gestiona recursos físicos:

- **Canchas:** Una Sede tiene múltiples Canchas.
- **Validación de Disponibilidad:** Antes de confirmar un partido, el sistema utiliza validarDisponibilidadCancha (en CompeticionService y Utils). Para que no se pueden juntar dos partidos en la misma cancha a la misma hora.

4. Reglas de Tiempo y Horarios

En el paquete Utils, la clase ValidadorHorario define reglas estrictas:

- **Horarios:** Tiene constantes HORA_APERTURA y HORA_CIERRE. El sistema impide programar partidos fuera de estas horas.
- **Validación:** El método validarDisponibilidadCancha() devuelve un booleano, para permitir o rechazar la creación de un partido.

5. Flujo de Control

- La Interfaz Gráfica (GUI) nunca habla directamente con los servicios o el modelo.

- Cuando el usuario hace click en "Guardar Jugador", la GUI le dice al GestorSistema, y este delega la tarea a JugadorService. Esto centraliza la lógica y protege los datos.

6. Persistencia y base de datos

- **Persistencia:** El sistema guarda el estado en un archivo binario. Esto asegura que si se cierra el programa, no se pierdan los datos existentes (los torneos, jugadores, etc).
- **Excepciones de Negocio:**
    - JugadorYaExisteException: Impide duplicados (mediante el DNI).
    - InscripcionException: impide inscribirse en un torneo cerrado o lleno.
    - equipoCompletoExeption: si hay 2 jugadores impide que se registre un tercero
    - EquipoYaExiste: evita que haya 2 equipos iguales

Arquitectura y Explicación de Clases

El sistema sigue una arquitectura en capas, separando claramente la vista, el controlador, el servicio (lógica de negocio) y el modelo.

A. Modelo (modelo)

Representa los datos y entidades del dominio.

- **Persona (Abstracta):** Clase base. Define atributos comunes como nombre, apellido y dni.
- **Jugador:** Hereda de Persona. Añade atributos como nivel, posición y estadísticas (partidosGanados, etc.).
- **Arbitro:** Hereda de Persona. Añade el número de licencia.
- **Equipo:** Agrupa una lista de 2 Jugadores. Contiene estadísticas del equipo.
- **Sede:** Representa un club. Contiene una lista de objetos Cancha.
- **Cancha:** Representa una pista física con atributos como superficie, iluminación y número.
- **Partido:** Representa un encuentro. Vincula dos Equipos, una Cancha, un Arbitro, un resultado y, opcionalmente, un Torneo. Ahora incluye duracionMinutos para validar horarios.
- **Torneo:** Gestiona una lista de Equipos inscritos y una lista de Partidos. Tiene un estado (PENDIENTE, EN_CURSO, FINALIZADO).

B. Servicios (servicios)

Aquí reside la lógica de negocio pura. Se utilizan interfaces para desacoplar la implementación.

- **Interfaces (IJugadorServicio, IEquipoServicio, etc.):** Definen el contrato de qué se puede hacer (registrar, eliminar, buscar) sin decir cómo.
- **JugadorServicio / ArbitroServicio / EquipoServicio:** Implementan la lógica CRUD (Crear, Leer, Actualizar, Borrar) de sus respectivas entidades. Validan duplicados antes de agregar a las listas en memoria.
- **CompeticionServicio:** Es el servicio más complejo. Gestiona Sedes, Torneos y Partidos. Se encarga de validar reglas de negocio complejas como la disponibilidad de canchas (delegando en un validador) y la búsqueda de partidos por fecha.

C. Controlador (controlador)

- **GestorSistema:** Es el **Controlador Principal (Facade)**. La vista solo habla con esta clase. El GestorSistema recibe las peticiones de la vista y las delega al servicio correspondiente (jugadorServicio, competicionServicio, etc.). También coordina la persistencia.

D. Persistencia (persistencia)

- **IPersistencia:** Interfaz que define guardarDatos y cargarDatos.
- **PersistenciaSerializable:** Implementación concreta que guarda todo el estado del sistema (listas de objetos) en un archivo binario (datos_padel.dat) usando la serialización de Java.

E. Utilidades (util)

- **Estilo:** Clase estática para centralizar colores, fuentes y bordes de la interfaz (Look & Feel).
- **ValidadorHorario:** Contiene la lógica matemática para verificar si dos rangos de horarios se superponen (usado para no reservar la misma cancha dos veces).
- **Ranking:** Lógica para ordenar listas de jugadores o equipos basándose en su ratio de victorias.
- **SelectorFecha:** Un componente gráfico personalizado (Dialog) para elegir fechas.

F. Vista (vista)

- **SistemaPadelGUI:** El JFrame principal. Configura el CardLayout para navegar entre pantallas.
- **PanelMenuPrincipal:** Muestra los botones para ir a otras secciones.
- **Paneles de Gestión (PanelGestionJugadores, etc.):** Contienen tablas (JTable) para ver datos y botones para abrir diálogos de registro/modificación.
- **PanelDisponibilidadCanchas:** Muestra visualmente qué canchas están ocupadas o libres en una fecha y hora (grilla de colores).

Dialogs y Excepciones:

El sistema maneja la interacción con el usuario y los errores de forma robusta.

A. Excepciones Personalizadas

Están en el paquete excepciones.

- **Dónde se usan?**
    - **InscripcionException:** Se lanza en Torneo.java cuando intentas inscribir un equipo y el torneo está lleno o el equipo ya está dentro.
    - **JugadorYaExisteException:** Se lanza en JugadorServicio.java y ArbitroServicio.java al intentar registrar un DNI duplicado.
- **Por qué?**
    - Para separar la **lógica** de la **vista**. El Servicio detecta el error (ej. "DNI repetido") pero no sabe cómo mostrarlo (no debe usar JOptionPane ahí). Lanza la excepción, y la Vista (el Panel) la captura (try-catch) y decide mostrar un mensaje de error visual. Esto permite que el mismo servicio pueda ser usado por una web o una app móvil en el futuro.

B. Dialogs (Ventanas Emergentes)

El sistema hace un uso extensivo de JOptionPane y JDialog.

1. **JOptionPane (Estándar):**
    - **Uso:** En todos los PanelGestion....
    - **Ejemplos:**
        - showConfirmDialog: Para mostrar formularios de entrada (ej. registrar jugador con varios campos) o confirmar eliminaciones.
        - showMessageDialog: Para notificar éxito ("Jugador guardado") o error (dentro de los catch de las excepciones).
        - showInputDialog: Para pedir un solo dato rápido, como un DNI para buscar.
2. **JDialog Personalizado:**
    - **Implementación:** util/SelectorFecha.java.
    - **Qué hace:** Es una clase que hereda de JDialog. Dibuja manualmente un calendario con botones para los días y navegación de meses.
    - **Por qué:** Java Swing estándar no tiene un componente de calendario nativo "bonito" (como JDatePicker de librerías externas). Esta clase implementa uno propio para no depender de librerías de terceros, demostrando cómo crear componentes modales complejos desde cero. Se usa en PanelDisponibilidadCanchas.

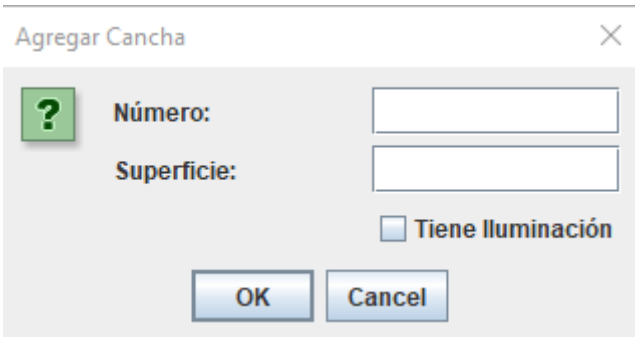
Interfaz gráfica

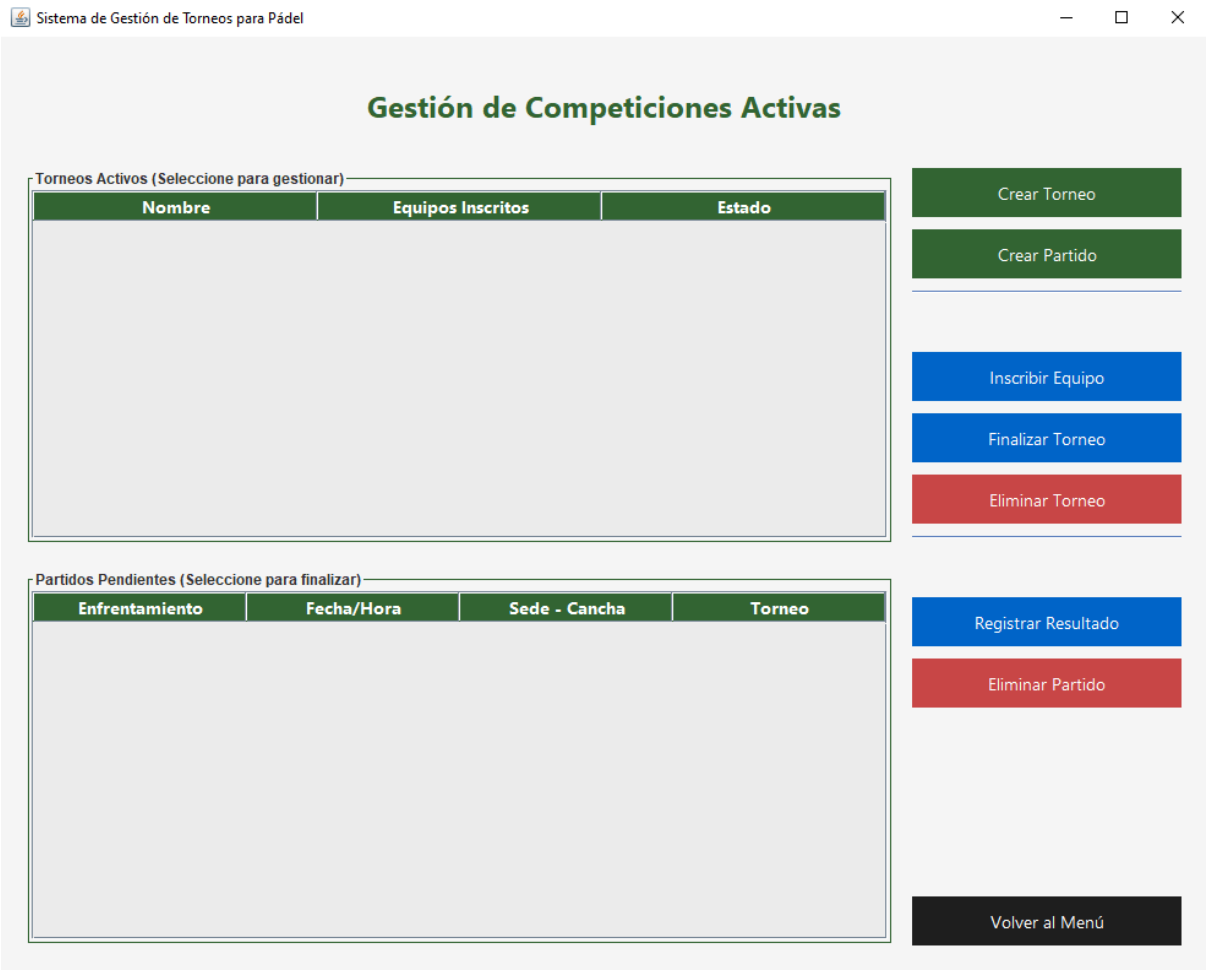
esta es la pestaña principal en la cual se pueden acceder a todas la funciones

estas son las pestañas de gestión de jugadores y equipos en las cuales se pueden agregar, modificar y eliminar jugadores, a la hora de registrar un jugador solicita el nombre y apellido, el dni, la posición y el nivel

luego está la pestaña de gestión de sedes y canchas la cual es similar a la de gestion de jugadores.

- A la hora de crear una nueva sede, se solicita el nombre de la sede y la locación.
- Para agregar canchas a la sede hay que hacer click en la sedes que ya están registradas y darle al botón agregar cancha, luego solicita el número de cancha y el tipo de superficie, además hay un checkbox el cual indica si la cancha posee o no iluminación

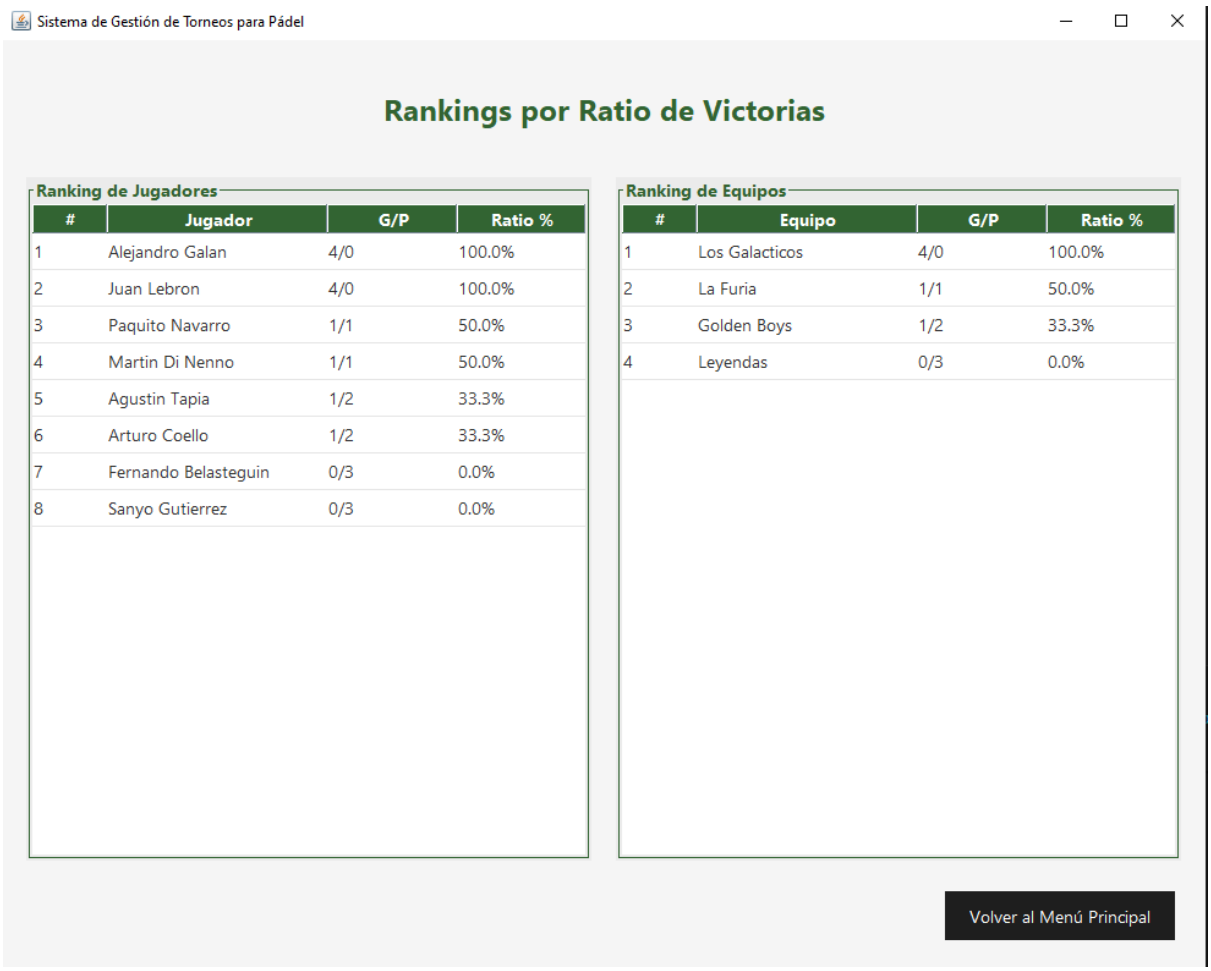




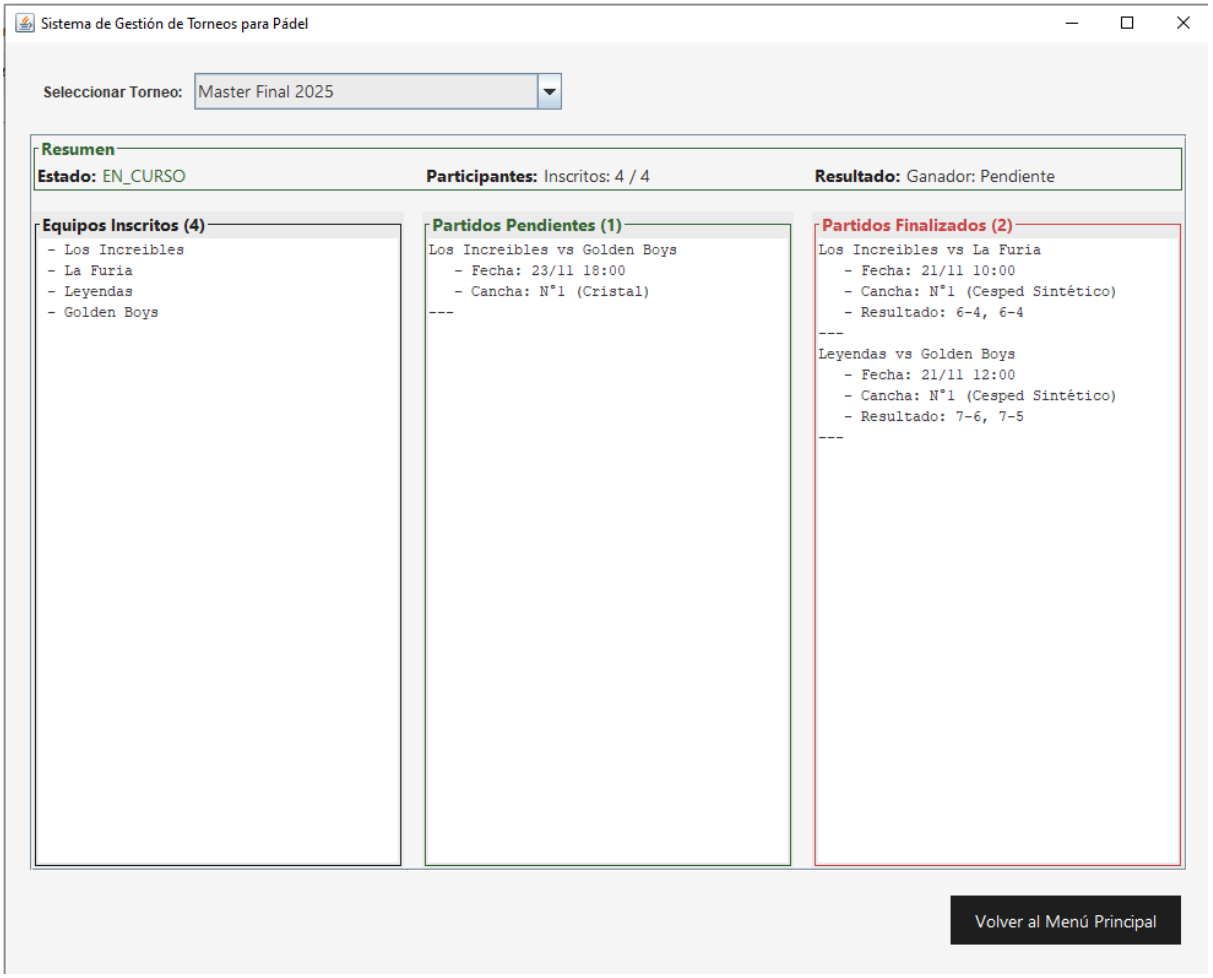
Por otro lado esta la gestion de competiciones activas la cual permite:

- crear torneo y partido
- gestionar los partidos pendientes: registrar como salieron los partidos y eliminarlos
- gestionar torneos activos: inscribir un equipo, finalizar el torneo y eliminarlo

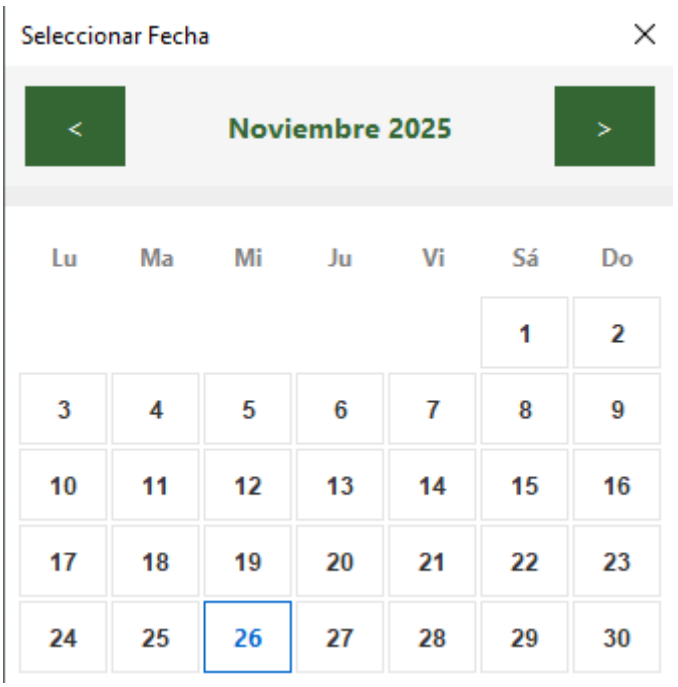
Esta tambien la interfaz de gestion de arbitros la cual es similar a la de gestionar jugadores



La interfaz por ratio de victorias la cual se divide en 2, en jugadores y en equipos



Esta la pestaña de historial de torneos en la cual quedan registrados los torneos pasados, los pendientes y los equipos que estan inscritos en los torneos en curso



Está la disponibilidad de canchas, en la cual se puede consultar que canchas estan disponibles segun el dia, la hora y la sede, cuando se agrega un torneo o partido la cancha se ocupa automáticamente

Identificación de patrones

Patrones de Diseño Implementados (SOLID y GRASP)

El código muestra un buen uso de patrones para mantener el software mantenible.

A. Patrones SOLID

1. **SRP (Single Responsibility Principle - Principio de Responsabilidad Única):**
    - **Ejemplo:** ValidadorHorario. Su única responsabilidad es comprobar si hay solapamiento de horas. No sabe de interfaces gráficas ni de guardar datos.
    - **Ejemplo:** PersistenciaSerializable solo se encarga de escribir/leer archivos, no de la lógica del torneo.
2. **OCP (Open/Closed Principle - Abierto/Cerrado):**
    - **Ejemplo:** El GestorSistema depende de la interfaz IPersistencia. Si mañana quieres guardar en una base de datos SQL, creas una clase PersistenciaSQL que implemente la interfaz y no tienes que tocar el código del GestorSistema, solo inyectar la nueva clase.
3. **LSP (Liskov Substitution Principle - Principio de Sustitución de Liskov):**
    - **Ejemplo:** Jugador y Arbitro pueden ser usados en cualquier lugar donde se espere una Persona (aunque en este sistema se usan mayormente de forma específica, la herencia respeta el contrato).
4. **ISP (Interface Segregation Principle - Segregación de Interfaces):**
    - **Ejemplo:** Las interfaces de servicio están separadas (IJugadorServicio, IEquipoServicio). El PanelGestionJugadores solo necesita métodos de jugadores, no necesita saber nada sobre los métodos de ICompeticionServicio.
5. **DIP (Dependency Inversion Principle - Inversión de Dependencias):**
    - **Ejemplo:** GestorSistema depende de interfaces (IJugadorServicio, IPersistencia) y no de las clases concretas directamente en su definición de tipos. Las implementaciones se inyectan o instancian en el constructor, permitiendo flexibilidad.

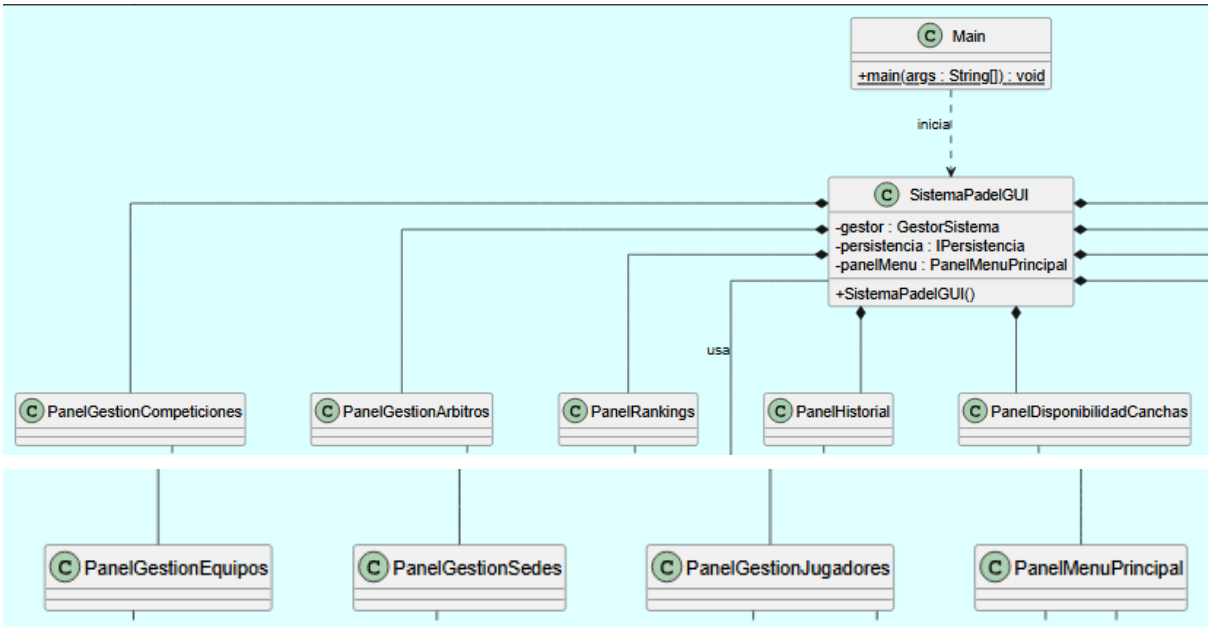
B. Patrones GRASP

1. **Controller (Controlador):**
    - **Implementación:** GestorSistema. Actúa como un intermediario entre la UI y la lógica interna. Recibe los eventos de la UI (ej. "botón registrar pulsado") y coordina la respuesta.
2. **Information Expert (Experto en Información):**
    - **Implementación:** Ranking. Es quien tiene la información de cómo calcular ratios y ordenar, por lo tanto, se le asigna la responsabilidad de ordenar las listas.
    - **Implementación:** Torneo. Sabe cuántos equipos tiene inscritos y cuál es su máximo, por lo tanto, es responsable de validar si cabe uno más (inscribirEquipo).
3. **High Cohesion (Alta Cohesión):**

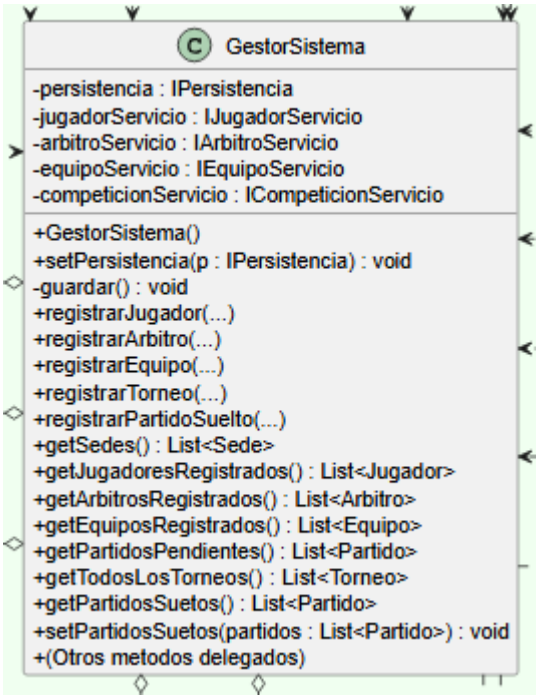
    - Las clases están agrupadas lógicamente. JugadorServicio solo trata con jugadores. No intenta gestionar canchas. Esto facilita entender qué hace cada clase.
4. **Low Coupling (Bajo Acoplamiento):**
    - El uso de interfaces en los servicios y en la persistencia reduce la dependencia directa entre clases concretas. La Vista no accede directamente a la lista de jugadores (ArrayList), sino que pide datos al Gestor.

UML (los getters y setters están incluidos)

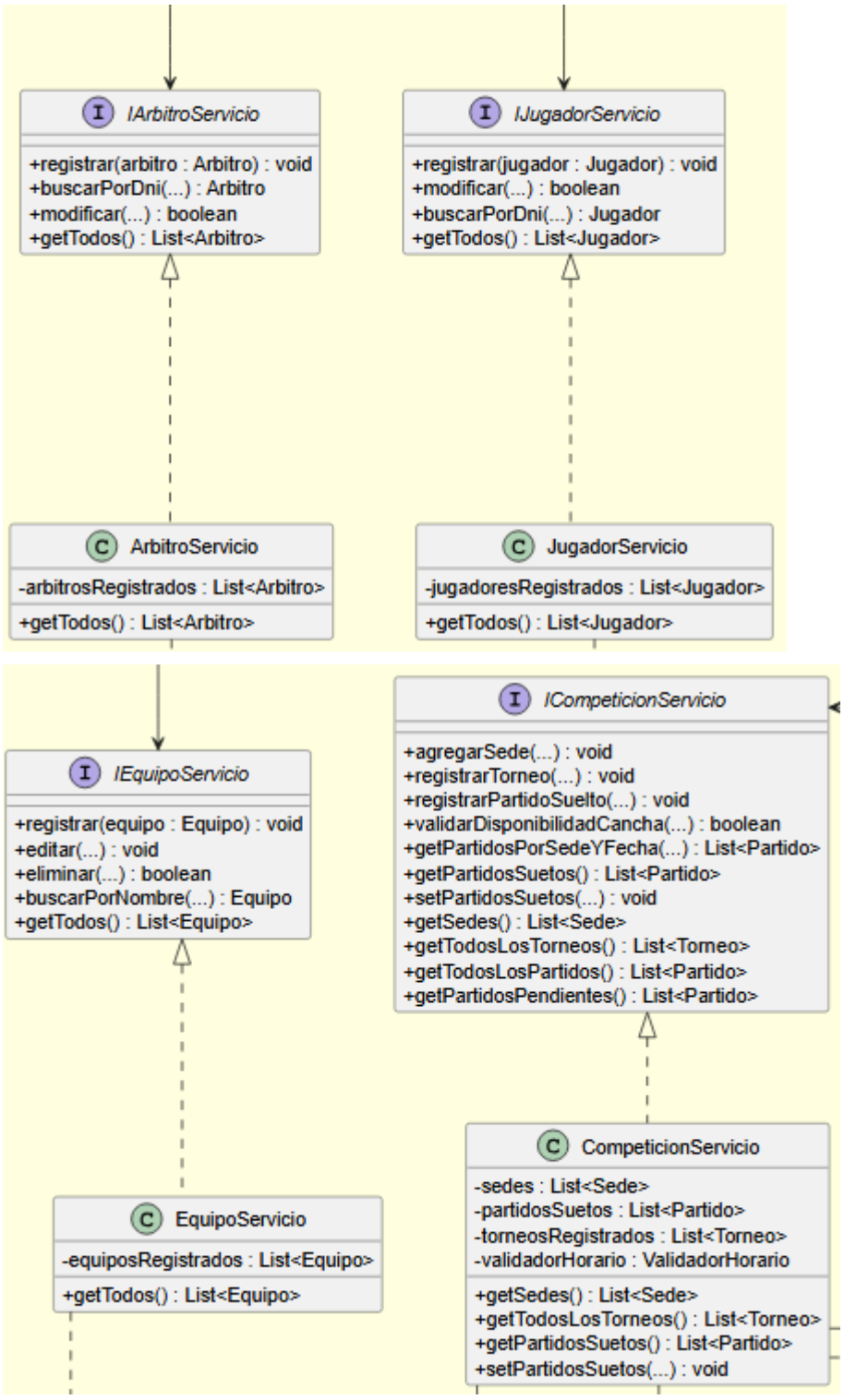
capa vista(GUI)



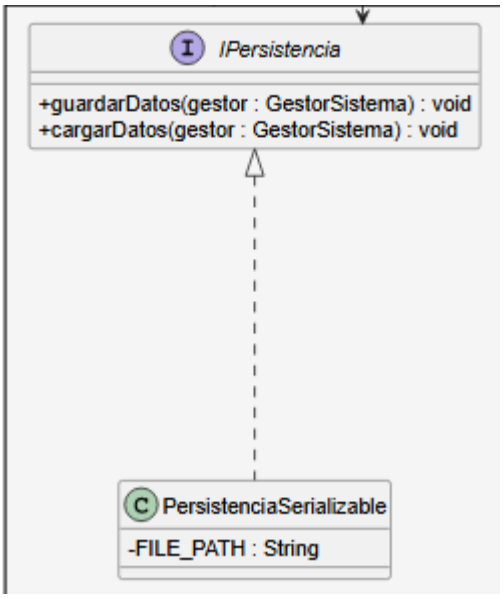
Capa control



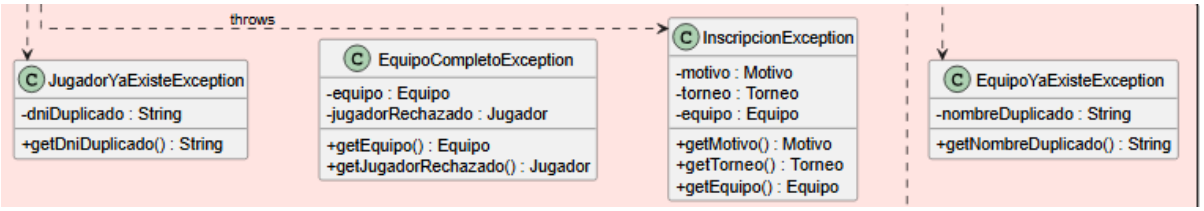
Capa de servicios



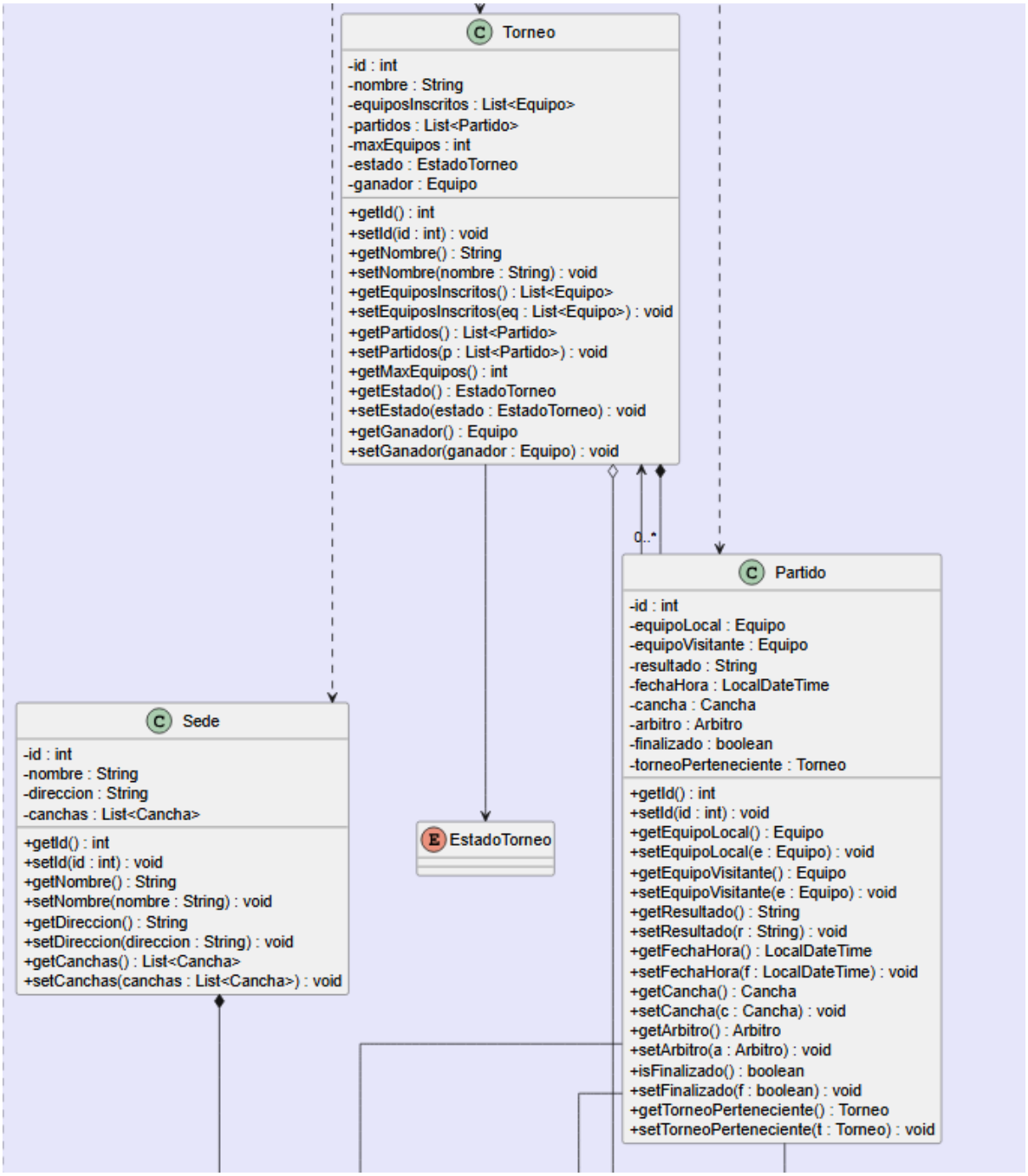
Capa persistencia

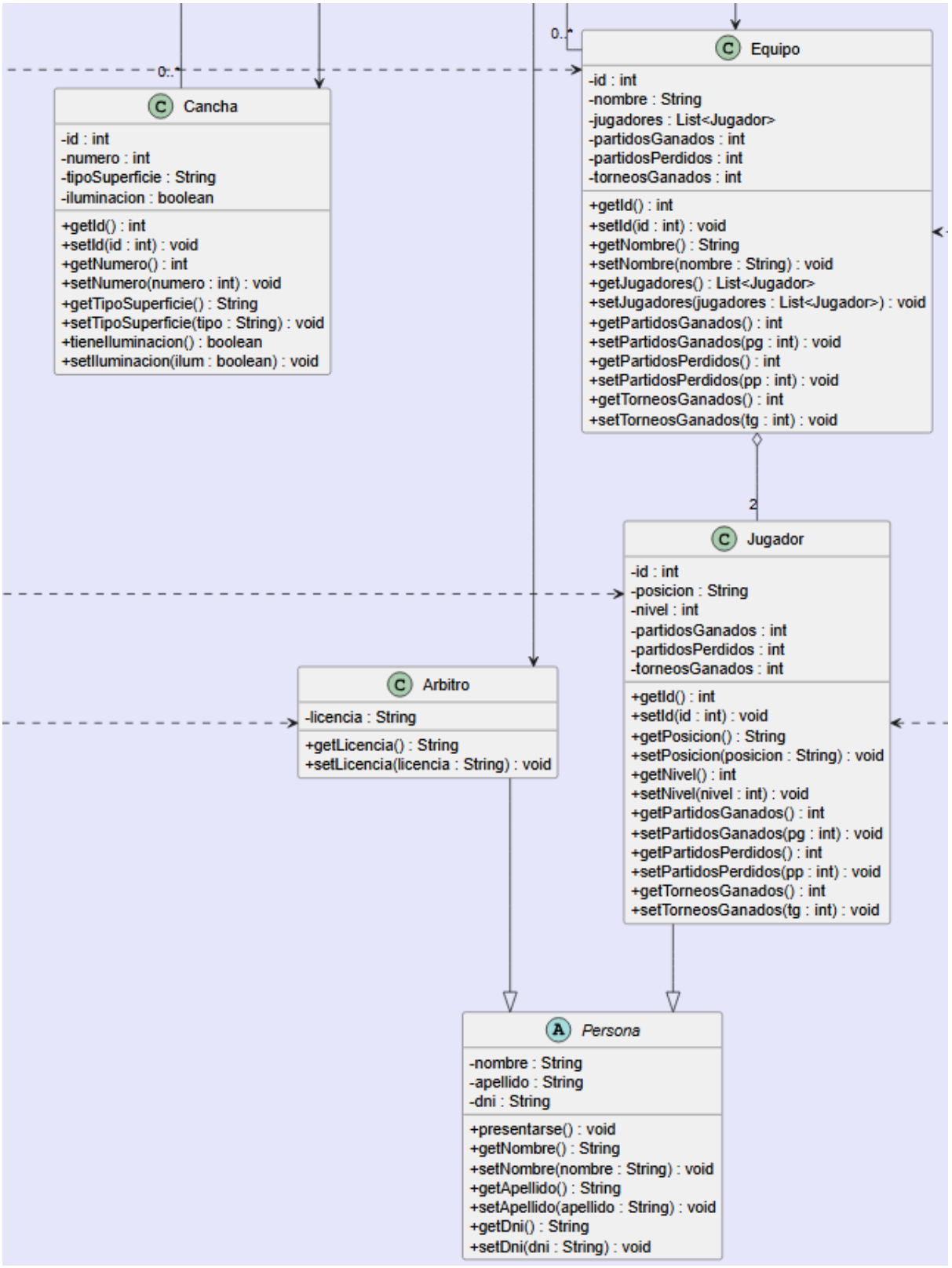


Capa de excepciones

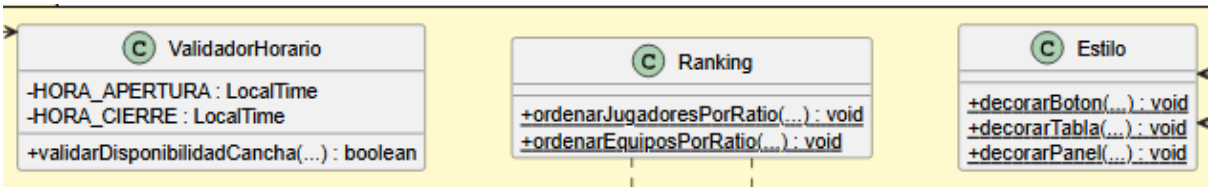


Capa Modelo


Torneo
-id : int
-nombre : String
-equiposInscritos : List<Equipo>
-partidos : List<Partido>
-maxEquipos : int
-estado : EstadoTorneo
-ganador : Equipo
+getId() : int
+setId(id : int) : void
+getNombre() : String
+setNombre(nombre : String) : void
+getEquiposInscritos() : List<Equipo>
+setEquiposInscritos(eq : List<Equipo>) : void
+getPartidos() : List<Partido>
+setPartidos(p : List<Partido>) : void
+getMaxEquipos() : int
+getEstado() : EstadoTorneo
+setEstado(estado : EstadoTorneo) : void
+getGanador() : Equipo
+setGanador(ganador : Equipo) : void
0..*
Partido
-id : int
-equipoLocal : Equipo
-equipoVisitante : Equipo
-resultado : String
-fechaHora : LocalDateTime
-cancha : Cancha
-arbitro : Arbitro
-finalizado : boolean
-torneoPerteneciente : Torneo
+getId() : int
+setId(id : int) : void
+getEquipoLocal() : Equipo
+setEquipoLocal(e : Equipo) : void
+getEquipoVisitante() : Equipo
+setEquipoVisitante(e : Equipo) : void
+getResultado() : String
+setResultado(r : String) : void
+getFechaHora() : LocalDateTime
+setFechaHora(f : LocalDateTime) : void
+getCancha() : Cancha
+setCancha(c : Cancha) : void
+getArbitro() : Arbitro
+setArbitro(a : Arbitro) : void
+isFinalizado() : boolean
+setFinalizado(f : boolean) : void
+getTorneoPerteneciente() : Torneo
+setTorneoPerteneciente(t : Torneo) : void
Sede
-id : int
-nombre : String
-direccion : String
-canchas : List<Cancha>
+getId() : int
+setId(id : int) : void
+getNombre() : String
+setNombre(nombre : String) : void
+getDireccion() : String
+setDireccion(direccion : String) : void
+getCanchas() : List<Cancha>
+setCanchas(canchas : List<Cancha>) : void
EstadoTorneo


0..*
0..*
Cancha
-id : int
-numero : int
-tipoSuperficie : String
-iluminacion : boolean
+getId() : int
+setId(id : int) : void
+getNumero() : int
+setNumero(numero : int) : void
+getTipoSuperficie() : String
+setTipoSuperficie(tipo : String) : void
+tieneIluminacion() : boolean
+setIluminacion(ilum : boolean) : void
Equipo
-id : int
-nombre : String
-jugadores : List<Jugador>
-partidosGanados : int
-partidosPerdidos : int
-torneosGanados : int
+getId() : int
+setId(id : int) : void
+getNombre() : String
+setNombre(nombre : String) : void
+getJugadores() : List<Jugador>
+setJugadores(jugadores : List<Jugador>) : void
+getPartidosGanados() : int
+setPartidosGanados(pg : int) : void
+getPartidosPerdidos() : int
+setPartidosPerdidos(pp : int) : void
+getTorneosGanados() : int
+setTorneosGanados(tg : int) : void
2
Jugador
-id : int
-posicion : String
-nivel : int
-partidosGanados : int
-partidosPerdidos : int
-torneosGanados : int
+getId() : int
+setId(id : int) : void
+getPosicion() : String
+setPosicion(posicion : String) : void
+getNivel() : int
+setNivel(nivel : int) : void
+getPartidosGanados() : int
+setPartidosGanados(pg : int) : void
+getPartidosPerdidos() : int
+setPartidosPerdidos(pp : int) : void
+getTorneosGanados() : int
+setTorneosGanados(tg : int) : void
Arbitro
-licencia : String
+getLicencia() : String
+setLicencia(licencia : String) : void
Persona
-nombre : String
-apellido : String
-dni : String
+presentarse() : void
+getNombre() : String
+setNombre(nombre : String) : void
+getApellido() : String
+setApellido(apellido : String) : void
+getDni() : String
+setDni(dni : String) : void

utils


ValidadorHorario
-HORA_APERTURA : LocalTime
-HORA_CIERRE : LocalTime
+validarDisponibilidadCancha(...) : boolean
Ranking
+ordenarJugadoresPorRatio(...) : void
+ordenarEquiposPorRatio(...) : void
Estilo
+decorarBoton(...) : void
+decorarTabla(...) : void
+decorarPanel(...) : void