Facultad de Ingeniería
Universidad de Concepción

# Desarrollo orientado a Objetos:

# Sistema de gestión de torneos


**Grupo: Proyecto 7**

**Integrantes: Ariel Fernández Fuentealba**

**Kurt Koresak Ortiz**

**Benjamín López Hermosilla**

**Docente: Geoffrey Jean-Pierre Christophe Hecht**

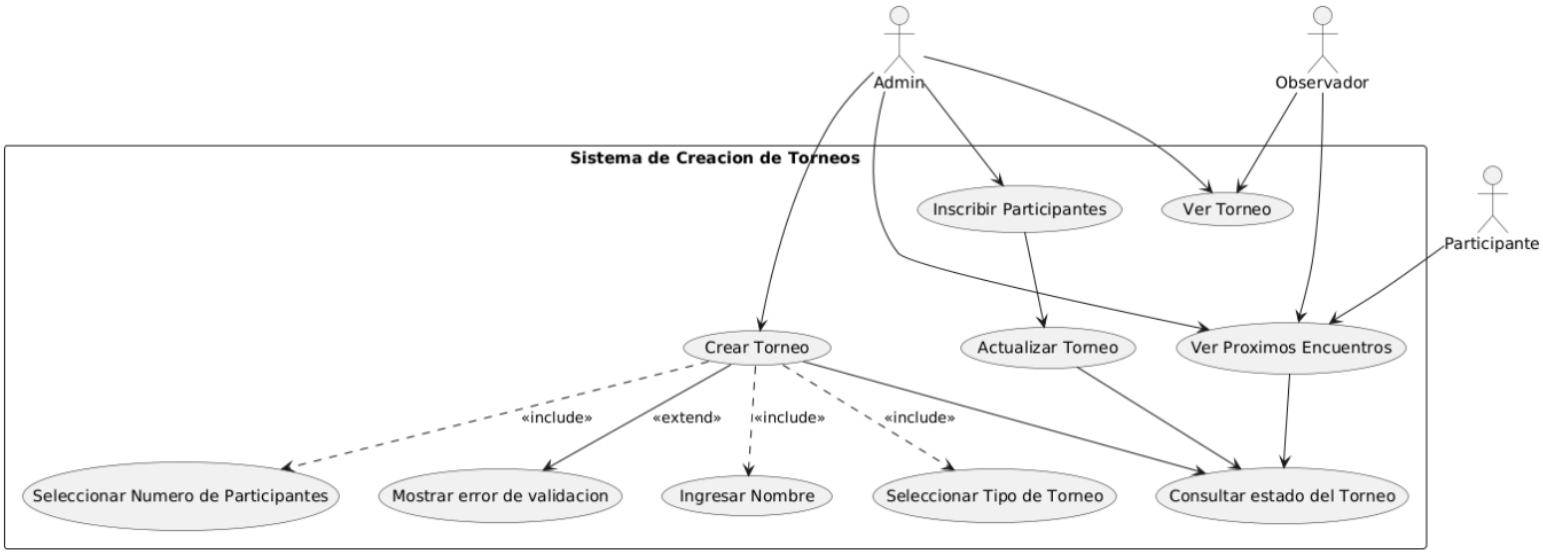
## Enunciado: Sistema de Gestión de Torneos

Este sistema está diseñado para facilitar la organización de torneos deportivos o de juegos. Permitirá a un organizador definir las características del torneo, como el nombre, la disciplina (ej. fútbol, ajedrez, videojuegos), las fechas y un formato principal (como eliminatoria directa, eliminatorio doble, liga simple...). Se podrán inscribir participantes, ya sean jugadores individuales o equipos, almacenando información básica como nombres y datos de contacto. El sistema deberá ser capaz de generar un calendario de enfrentamientos o un bracket inicial basado en los inscritos y el formato. Durante el torneo, se registrarán los resultados de cada enfrentamiento, lo que actualizará automáticamente las posiciones, el avance en el bracket o las tablas de clasificación. Los usuarios podrán visualizar el estado actual del torneo, los próximos encuentros y las estadísticas generales.

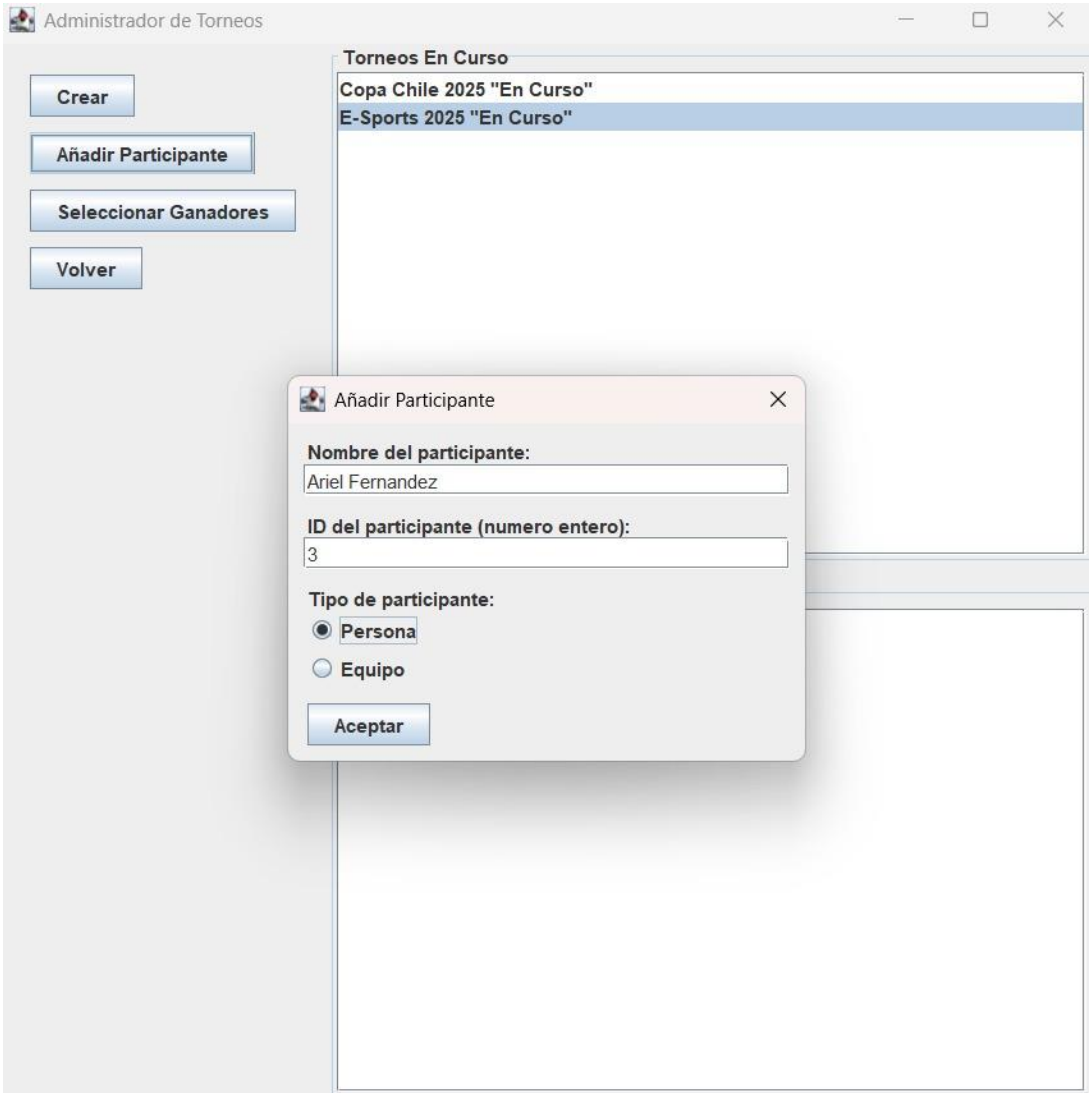
A continuación, iremos presentando lo más relevante de nuestra propuesta de solución.

### 1) Diagrama de casos:

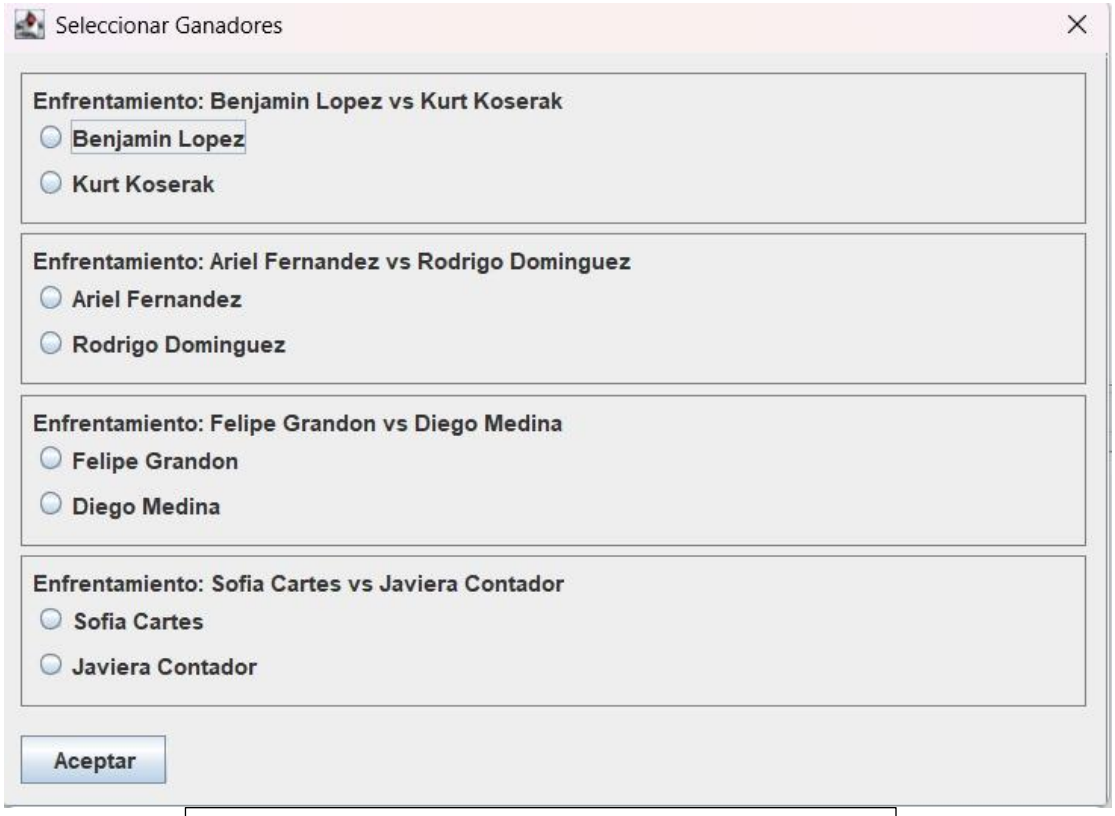
Gráficamente, el programa está pensado para que funcione desde 2 perspectivas (tanto como administrador de un torneo, como un espectador de tal), lo cuál está ejemplificado en el siguiente diagrama de casos:



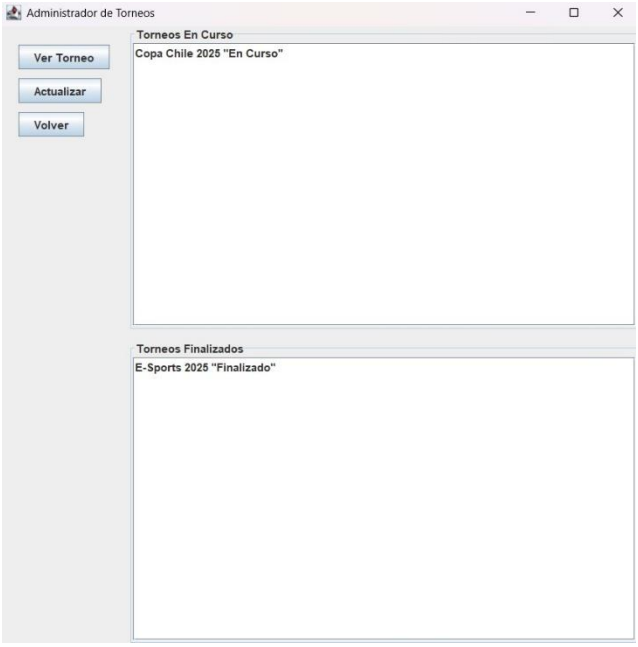
## 2) Interfaces:



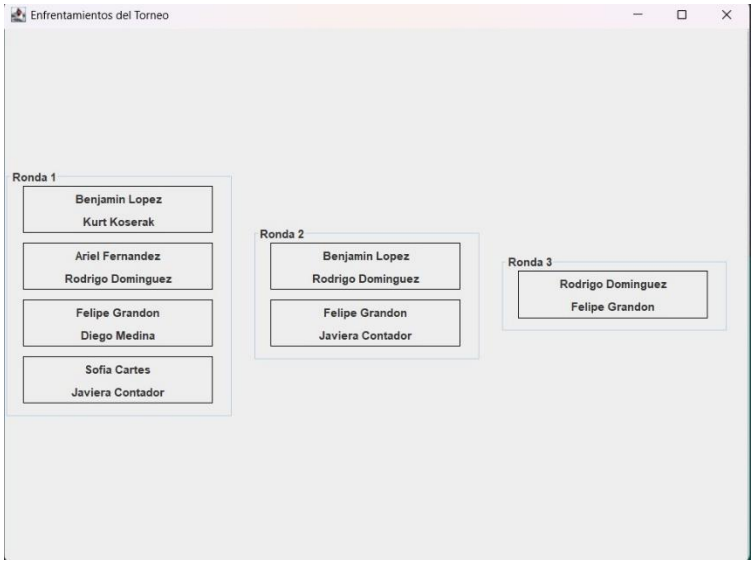

**Adición de un participante a un torneo**




**Perspectiva del administrador**

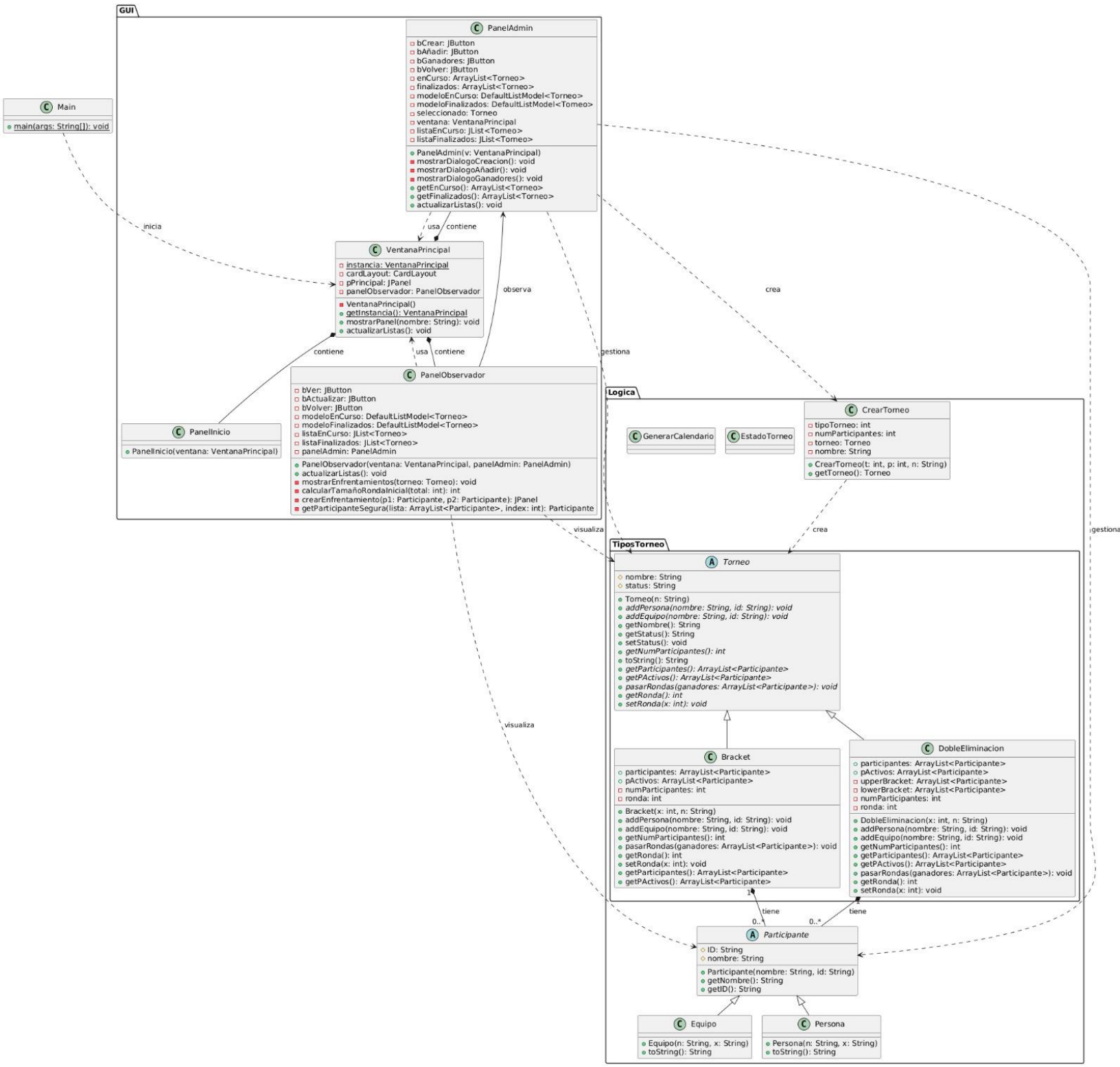



**Panel observador**




**Torneo Graficado**

# 3) Diagrama UML



## 4) Patrones de diseño aplicados

A lo largo del desarrollo de este proyecto consideramos prudente utilizar 2 patrones de diseño para la generalización del código:

### 3.1) Singleton – VentanaPrincipal

- **Propósito:** Garantizar que solo exista una única ventana principal en ejecución Y nos permite evitar errores por múltiples ventanas activas y centralizar la navegación entre paneles. Al final, la GUI es el corazón del sistema, por lo que tener una sola instancia asegura estabilidad y menor uso de recursos.
- **Clases implicadas:**
    - VentanaPrincipal: Implementa el patrón.

### 3.2) Factory – CrearTorneo

- **Propósito:** Centralizar la lógica de creación de torneos según el tipo seleccionado. Con esto evitamos que otras partes del programa necesiten conocer o gestionar directamente las subclases de Torneo. Así, extraemos la lógica switch a una clase TorneoFactory, aplicando el principio de responsabilidad única y facilitando una futura extensión sin necesidad de una modificación de esta clase.
- **Clases implicadas:**
    - CrearTorneo: actúa como fábrica.
    - Torneo: clase base o abstracta.
    - Bracket, Liguilla, DobleEliminacion: subclases concretas.

## 5) Decisiones importantes de diseño

Para mayor limpieza del código hemos decidido separar responsabilidades en paquetes (GUI, Logica, Participantes, TiposTorneo), a su vez, facilitamos el mantenimiento a futuro del código. Singleton tomó un rol protagónico a la hora de generar la ventana principal, evitamos conflictos visuales y múltiples interfaces (entre otros conflictos). La implementación de una fábrica simple (CrearTorneo) también fue un acierto para controlar el tipo de torneo creado dinámicamente, y no precisamente por la necesidad de usar un patrón, sino más bien el seguir este molde nos permitió conocer el posible potencial y extensibilidad que puede tener nuestro proyecto (aún para muchos torneos en simultaneo, solo cambiaría el rol que actualmente cumple Singleton). Y como adicional, se priorizó la visualización del estado del torneo a través de paneles específicos tanto para observador como para el administrador.

## 6) Problemas encontrados y autocrítica

Es de reconocer que dificultad inicial (y la mayor del proyecto) fue hallar un punto de partida sin haber tenido nada, tanto para organizar la lógica de los torneos y la imposibilidad de tener una idea visual de esto antes de tener la lógica. Otra de las falencias que hemos encontrado ha sido el problema de la persistencia de datos (todo se pierde al cerrar el programa, lo cual es perfectamente enmendable con un checkpoint en un bloc de notas que le diga al programa en que fue lo último que quedó). También, lo que dificultó el desarrollo en conjunto fue la escasa documentación al inicio del desarrollo, lo cual se mejoró luego de los feedbacks realizados por nuestro referente. Y por último, durante el desarrollo de la implementación se llegó a notar repetición de lógica en algunos puntos (por ejemplo, cambios de paneles, impresión de resultados), lo cual si hubiese sido avistado con mayor antelación habría ahorrado tiempo considerable.

Todo esto nos lleva a ser críticos con nuestro grupo, como el hecho de haber implementado una estructura más formal de patrones desde el comienzo nos habría facilitado el orden y legibilidad del código (y la no repetitividad), lo que finalmente nos lleva a tener 2 aspectos a mejorar para una futura extensibilidad del código:

- Implementar persistencia (uso de archivos o base de datos) para guardar torneos y resultados.
- Documentar el proyecto desde etapas tempranas para tener un avance regulado por checkpoints (sin acarreos de errores).