Metodología y Programación Orientada a Objetos

Caso de Estudio

Integrante:

- Gabriel Alejandro García Angulo
- Roderick Uziel Caldera Torrez
- Nicolás Rafael Laguna Vallejos
- Julio César Salamanca Ortiz
- Farid Eduardo Zuniga Rico

Docente:

José Durán García

Managua, 19 de Septiembre del 2025

La asociación en UML es la relación estructural más general entre dos clases y expresa que una “conoce/usa” a la otra durante su vida útil. En el diagrama se representa con una línea simple que puede llevar nombre, roles en cada extremo y multiplicidades. Es flexible: permite indicar direccionalidad (navegabilidad), atributos de la propia relación y calificaciones. En términos de diseño, conviene modelarla cuando dos conceptos colaboran sin dependencia de ciclo de vida; por ejemplo, una clase Partida que “usa” un Árbitro para supervisión sigue existiendo aunque se cambie el árbitro. Este tipo de vínculo es la base desde la cual se especializan la agregación y la composición.

La agregación es una forma particular de asociación “todo-parte” débil que se dibuja con un rombo blanco en el lado del “todo”. Describe que las partes pueden existir de manera independiente y, a menudo, ser compartidas o reasignadas. Por ello no impone dependencia de ciclo de vida: si el todo desaparece, las partes no tienen por qué hacerlo. En el dominio de eSports, un Equipo agrega Jugador porque los jugadores pueden seguir inscritos en la plataforma o incluso pasar a otro equipo cuando este se elimina o se disuelve. Modelar así ayuda a reflejar reglas de negocio como transferencias y agentes libres, y evita acoplar la vida del jugador a la del equipo.

La composición es también una relación “todo-parte” pero fuerte: se representa con rombo negro y connota propiedad exclusiva y coincidencia de ciclo de vida. La parte no existe fuera del todo y su destrucción va unida a la del compuesto. Se usa cuando una entidad gestiona, crea y elimina a sus partes, garantizando invariantes internas. Para el caso, un Torneo compone sus Partida: una partida pertenece a un torneo específico y si el torneo se

elimina, sus partidas también desaparecen. Esta decisión de modelado impide partidas "huérfanas" y hace explícito que el calendario y resultados están subordinados al torneo.

La herencia (generalización/especialización) modela la relación "es-un". Permite que una subclase herede atributos y operaciones de su superclase y añada o redefina comportamiento. Se representa con un triángulo hueco apuntando a la superclase. Además de factorizar código, guía el diseño hacia polimorfismo y sustitución. En este dominio, Jugador y Árbitro pueden especializar una superclase Persona para compartir la identidad nominal, y a la vez mantener atributos propios (como alias y ranking para Jugador). Es una decisión de diseño que mejora claridad y reutilización, sin alterar las reglas del negocio.

La multiplicidad (o cardinalidad) precisa cuántas instancias de una clase pueden vincularse con cada instancia de la otra y se anota en cada extremo de la asociación (por ejemplo, 1, 0..1, 0..*, 1..*). En la práctica, define restricciones del modelo y condiciona la implementación (colecciones vs. referencias únicas, validaciones y claves en persistencia). Los tres patrones más habituales son: 1:1, cuando a cada instancia de A corresponde exactamente una de B; 1:N, cuando una instancia de A se asocia con cero o muchas de B pero cada B solo con una A; y N:M, donde varias instancias de A se asocian con varias de B. En diseño lógico de bases de datos, el N:M suele resolverse con una clase/tabla intermedia (por ejemplo, una inscripción o participación).

Análisis al caso del Sistema de Gestión de Torneos de eSports.

El dominio exige modelar siete entidades: Torneo, Partida, Equipo, Jugador, Juego, Categoría y Árbitro. Para reflejar el ciclo de vida, Torneo compone Partida porque las partidas nacen y mueren con el torneo. Cada Partida se asocia con exactamente dos Equipo (dos enlaces con rol, p. ej. equipoA y equipoB), con un Árbitro y con un Juego. Juego se asocia a una Categoría en relación 1:N (una categoría puede clasificar muchos juegos, pero cada juego pertenece a una sola categoría); la categoría existe por separado del juego. Equipo mantiene una agregación 1:N con Jugador: los jugadores continúan existiendo si el equipo se elimina y pueden migrar. Finalmente, Torneo y Equipo se relacionan N:M porque un equipo puede participar en varios torneos y un torneo admitir múltiples equipos; esto se implementa con colecciones en ambos lados o con una entidad de unión (p. ej., Inscripción) si hiciera falta registrar estado adicional (fecha de alta, seed, etc.). Con herencia, Persona sirve como supertipo para Jugador y Árbitro, factoriza el atributo nombre y facilita validaciones coherentes. Estas decisiones hacen que el diagrama sea fiel al caso, y además guían una implementación Java limpia: composición con listas internas y borrado controlado en Torneo, agregación con listas no propietarias en Equipo, validaciones de multiplicidad en constructores/métodos (p. ej., impedir que Partida reciba el mismo equipo en ambos extremos) y una relación N:M consistente mediante métodos espejo o una entidad de unión si el sistema evoluciona.

Referencias:

Abad, P., & Sánchez, A. (2015). *Modelos de datos: Lenguaje UML. Cardinalidad o multiplicidad* [PDF]. https://gbif.es/wp-content/uploads/2018/01/Modelos-de-datos-Lenguaje-UML-Paloma-Abad-y-Alejandra-Sanchez.pdf

Barquinero, J. (2019). *Agregación Vs Composición en diagramas de clases. UML*. Blog de SEAS. https://www.seas.es/blog/informatica/agregacion-vs-composicion-en-diagramas-de-clases-uml/

Lalindri. (2023). *Relaciones de diagrama de clases UML explicadas con ejemplos*. Creately Blog. https://creately.com/blog/es/diagramas/relaciones-de-diagrama-de-clases-uml-explicadas-con-ejemplos/

Lucid Software. (2025). Tutorial de diagrama de clases UML. https://www.lucidchart.com/pages/es/tutorial-de-diagrama-de-clases-uml

Miro. (s. f.). *Diagrama de clases UML: Qué es, cómo hacerlo y ejemplos*. https://miro.com/es/diagrama/que-es-diagrama-clases-uml/