Universidad Americana

Facultad de Ingeniería y Arquitectura

Metodología y programación orientada a objetos I - Grupo 1

Caso de Estudio - Diagrama de Clases

Integrantes Grupo 5

Priscila Julieth Selva Flores

Jenny Lomary Orozco Chávez

Emma Cecilia Serrano Urroz

Francisco Javier Guevara Arauz

Jan Carlo Páramo Gutiérrez

Docente

José Durán García

Fecha 16/09/2025

Una organización quiere desarrollar un sistema para gestionar torneos de videojuegos (eSports). Este sistema se encargará de manejar equipos, jugadores, juegos, torneos y partidas.

Requisitos del sistema

- Un torneo está compuesto por varias partidas (matches). Si el torneo se elimina, sus partidas también.
- Cada partida se juega entre dos equipos y está asociada a un juego (por ejemplo, LoL, CS:GO, etc.).
- Un equipo está compuesto por varios jugadores. Si el equipo se elimina, los jugadores no se eliminan (pueden ser asignados a otro equipo).
- Cada jugador tiene un nombre, alias y ranking.
- Cada juego tiene una categoría (FPS, MOBA, etc.). Las categorías existen por separado del juego.
- Un árbitro supervisa cada partida. Un árbitro puede supervisar muchas partidas, pero una partida tiene un solo árbitro.
- Un equipo puede participar en varios torneos, y un torneo puede tener varios equipos.

Instrucciones de la Actividad

Investiga los tipos de relaciones en diagramas de clases UML.

Asociación: La Asociación es un término amplio que representa una conexión o relación lógica entre clases. Es el tipo de relación más común y se utiliza para mostrar dependencias semánticas. Se representa típicamente con una línea continua entre las clases involucradas. Las asociaciones pueden tener un nombre, roles en sus extremos, multiplicidad, visibilidad y otras propiedades (*Diagrama De Clases.*, S.f.). Puede relacionar cualquier número de clases (binaria entre dos, ternaria entre tres, etc.). Existen varios tipos de asociación:

- Bidireccional y Unidireccional: Son los tipos más comunes. Una asociación unidireccional se representa con una línea con una punta de flecha que indica el flujo direccional. En una asociación unidireccional, una clase conoce a la otra y puede llamar a sus métodos, pero no a la inversa.
- Reflexiva: Ocurre cuando una clase se relaciona consigo misma, es decir, puede tener múltiples roles o responsabilidades dentro de la misma clase.

- Agregación y Composición: Son especializaciones de la relación de asociación, que indican una relación de "parte-todo".

Un ejemplo de asociación simple sería "Una mascota pertenece a una persona", o una clase Vuelo asociada con una clase Avión de forma bidireccional.

Composición: La composición se utiliza para modelar una relación jerárquica donde una clase es "parte de" otra, con una dependencia fuerte en el ciclo de vida (Megino, 2013). Esto significa que la vida de la clase contenida (la "parte") está intrínsecamente ligada a la vida de la clase contenedora (el "todo" o "compuesto").

Las características clave de la composición incluyen:

- Dependencia del ciclo de vida: La supresión del objeto compuesto (el "todo") conlleva la supresión de sus componentes (las "partes"). Por ejemplo, si un centro de visitantes es demolido, su recepción y baños también se demolerían, ya que no pueden existir separados del centro.
- No compartición: Los componentes no pueden ser compartidos por varios objetos compuestos. Cada parte pertenece exclusivamente a un único todo.
- Acoplamiento estricto: Implica un acoplamiento más estricto y contención que otros tipos de asociaciones.
- Sentido de existencia: Los elementos que forman parte del todo no tienen sentido de existencia cuando el todo desaparece.

En UML, la composición se representa gráficamente con un diamante de color negro o rombo relleno colocado en el extremo de la clase que representa el "todo" (la clase contenedora), unido por una línea a la clase que representa la "parte".

Agregación: La Agregación es un tipo de relación en los diagramas de clases UML (Unified Modeling Language) que representa una forma de asociación entre clases, específicamente una composición débil o una relación de "parte-todo" (Megino, 2013). Se utiliza para modelar situaciones en las que un objeto es parte de otro, pero sus componentes pueden existir de forma independiente del "todo" o contenedor.

Características de la Agregación:

- Relación de "Parte-Todo": Indica que una clase es una colección o un contenedor de otras clases, donde los elementos forman parte de un objeto más grande.
- Independencia de los Componentes: La característica más distintiva es que el tiempo de vida de las clases contenidas no tiene una dependencia fuerte del tiempo de vida de la clase contenedora (el "todo"). Esto significa que los componentes no se destruyen automáticamente cuando desaparece la clase que los contiene. Por ejemplo, si una Biblioteca se disuelve, los Libros que la componían siguen existiendo. Otro ejemplo es que si un Profesor tiene Estudiantes y el profesor fallece, los estudiantes no mueren con él o ella.
- Compartición de Componentes: Los componentes pueden ser compartidos por varios objetos compuestos, ya sea de la misma asociación de agregación o de varias distintas. Por ejemplo, un motor (MTR01) puede ser parte de un coche (MC01), pero también podría ser parte de cualquier otro modelo de coche; si el coche MC01 se elimina, no es necesario eliminar el motor.
- Acoplamiento Débil: La relación de agregación implica un acoplamiento más débil en comparación con la composición.

En los diagramas de clases UML, la agregación se representa mediante un diamante de color blanco o rombo hueco que se coloca en el extremo de la clase que representa el "todo" (la clase contenedora o "padre"), unido por una línea a la clase que representa la "parte" (la clase contenida o "hija").

Herencia: Se utiliza para establecer una conexión lógica donde una clase adquiere las características y comportamientos de otra (*Diagramas UML*, 2023).

La herencia es un mecanismo que permite a una clase, conocida como clase hija o subclase (o también clase derivada), recibir y reaprovechar los atributos y métodos de otra clase, llamada clase padre o superclase (o clase base). Los atributos y métodos heredados se suman a los que la clase hija ya posee por sí misma.

Sus principales propósitos son:

- Reutilización de código: Permite aprovechar miembros de clases ya existentes, aprobadas y testeadas, sin necesidad de modificar el código original y sin romper su funcionalidad.

- Extensión de funcionalidad: Se puede tomar una clase existente y crear una nueva clase que herede su funcionalidad y añada lo específico y novedoso que se necesite.
- Generalización y Abstracción: Ayuda a agrupar características y comportamientos comunes en una superclase, evitando la repetición de código en las subclases. Si se cambia o agrega un atributo o método en la superclase, el cambio se aplica automáticamente a todas las subclases.
- Modelado conceptual: Simplifica la comprensión del sistema al mostrar jerarquías lógicas.

La herencia se aplica cuando una clase hija "es un" tipo especial o más específico de la clase padre. Es crucial diferenciarla de otras relaciones como la asociación ("tiene un" o "se compone de"), ya que aplicar herencia incorrectamente puede llevar a un mal diseño.
Por ejemplo:

- Un Auto es un Vehículo.
- Un Perro es una Mascota.

Sin embargo, una Persona tiene un Domicilio, no "es un" domicilio; en este caso, aplicaría una asociación, no herencia.

Multiplicidad / cardinalidad (1:1, 1:N, N:M)

La multiplicidad es una propiedad de las relaciones, especialmente las asociaciones, que indica cuántas instancias de una clase pueden estar relacionadas con cuántas instancias de otra clase (*Multiplicidad*, 2022). Permite comprender rápidamente las dependencias y relaciones entre objetos, clases y atributos en un sistema.

¿Dónde se indica?
Las notaciones de multiplicidad deben indicarse en cada extremo de la asociación. Se colocan cerca de los extremos de la línea que conecta las clases involucradas en la relación.

Multiplicidad y Cardinalidad (1:1, 1:N, N:M)
Estas cardinalidades tradicionales se expresan utilizando las notaciones de multiplicidad:

- 1:1 (Uno a uno):

Indica que una instancia de la Clase A se asocia con exactamente una instancia de la Clase B, y viceversa.

Se representa con 1 en ambos extremos de la asociación.
Ejemplo: En un sistema de carrito de compras, cada Pedido tiene una y solo una Información del Pedido, y la Información del Pedido pertenece a uno y solo un Pedido.

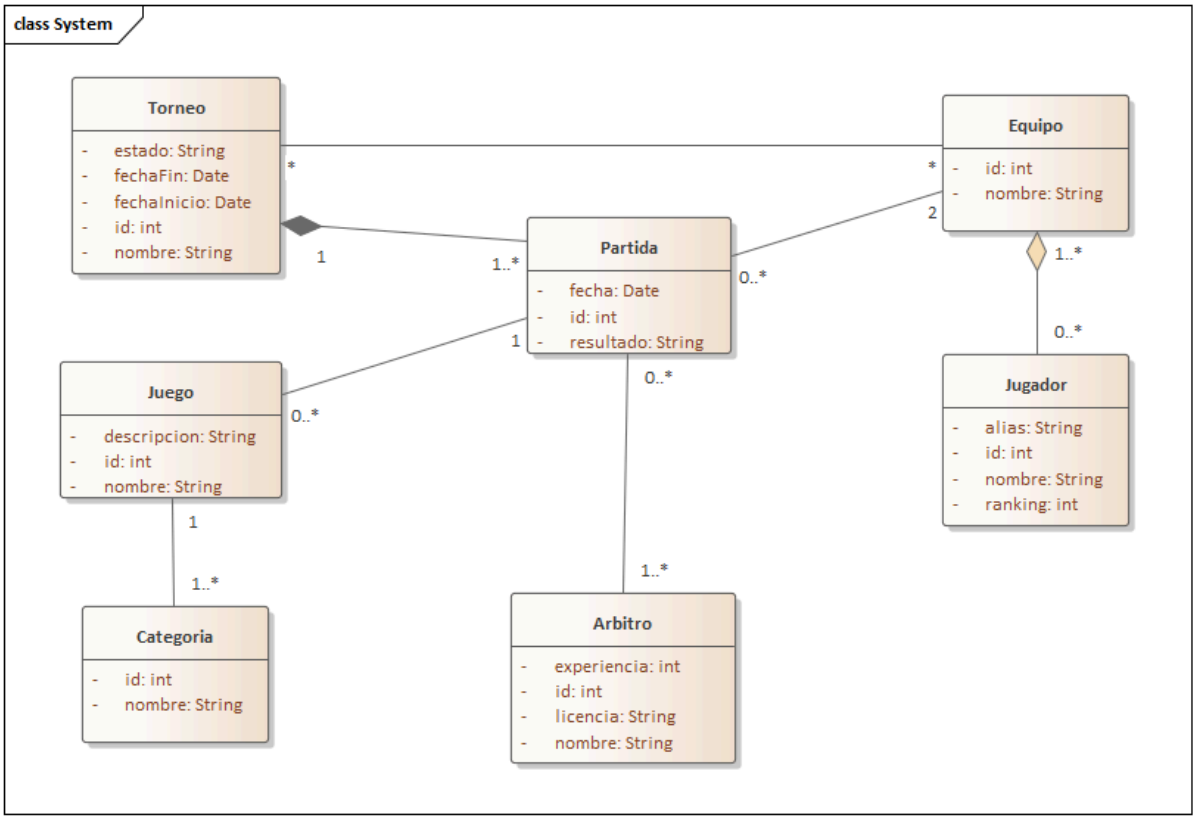
- 1:N (Uno a muchos):

Indica que una instancia de la Clase A puede asociarse con una o muchas instancias de la Clase B, pero cada instancia de la Clase B se asocia con una sola instancia de la Clase A.
Se representa con 1 en el extremo de la Clase A y 1..* o 0..* en el extremo de la Clase B.
Ejemplo: Una Empresa tendrá uno o más empleados (1..*), pero cada Empleado trabaja para una sola Empresa (1).
Ejemplo: Un Cliente puede tener cero o muchos pedidos (0..*), pero un Pedido puede pertenecer a un solo Cliente (1).

- N:M (Muchos a muchos):

Indica que muchas instancias de la Clase A pueden asociarse con muchas instancias de la Clase B, y viceversa.
Se representa con * o 1..* en ambos extremos de la asociación.
La referencia a "multiplicidad muchos a muchos" en el contexto de la agregación o composición ilustra cómo este concepto se aplica para definir relaciones complejas donde los componentes pueden ser compartidos.

Analiza el caso de estudio del Sistema de Gestión de Torneos de eSports y elabora un diagrama de clases UML donde representes las entidades (Torneo, Partida, Equipo, Jugador, Juego, Categoría, Árbitro), sus atributos y relaciones.



Referencias

Diagrama de clases. Teoria y ejemplos. (n.d.). diagramas UML. Recuperado de: https://diagramasuml.com/diagrama-de-clases/

Megino, J. M. (2013, Enero 25). *Agregación Vs Composición en diagramas de clases. UML. | Blog SEAS*. SEAS, Estudios Superiores Abiertos. Recuperado de: https://www.seas.es/blog/informatica/agregacion-vs-composicion-en-diagramas-de-clases-uml/

Que Es Multiplicidad. (2022, Febrero 9). Visual Paradigm Blog. Recuperado de: https://blog.visual-paradigm.com/es/what-is-multiplicity/

Relaciones de diagramas de clases en UML explicadas con ejemplos. (2023, Febrero 6). Creately. Recuperado de: https://creately.com/blog/es/diagramas/relaciones-de-diagrama-de-clases-uml-explicadas-con-ejemplos/