

Tema 1. Domain Driven Design

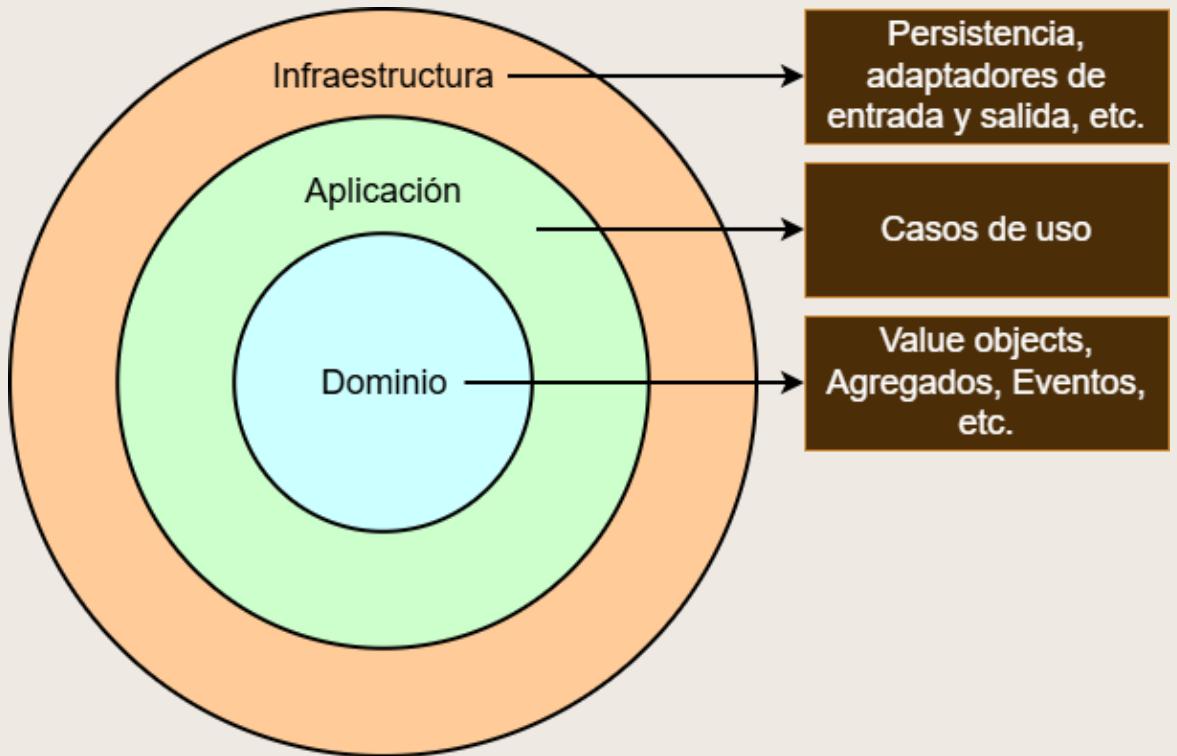


UNIVERSIDAD
DE MURCIA

ATICA

¿Qué es Domain Driven Design (DDD)?

Es una forma de diseñar software poniendo el dominio del negocio en el centro de la aplicación.



¿Qué necesitamos para aplicar DDD?

- Utilizar un **lenguaje ubicuo**.
- Establecer un **contexto** en el que se aplicará.
- Un **experto del dominio**.



Elementos de DDD

Disponemos de varios tipos de elementos para modelar nuestro dominio:

- Value Object
- Agregados
- Agregado raíz



Elementos de DDD: Value Objects

Es una pequeña unidad que representa una característica o valor de un concepto de nuestro dominio.

- **Inmutable**

Si necesitamos modificarlo, crearemos uno nuevo.

- **Siempre será válido**

En el momento de su creación se verificará que cumple las reglas de negocio.

- **No tiene identidad**

No tiene identificador único.

- **Encapsulado**

Agrupa datos y comportamiento, reforzando el principio de responsabilidad única.



Elementos de DDD: Value Objects

Algunos ejemplos:

- Correo electrónico
Se valida en la creación y se compara por su valor.
- Identificador de usuario
Encapsula el tipo y el número. No tiene identidad propia.
- Rango de fechas
Define reglas como inicio \leq fin.



Elementos de DDD: Agregados

Es un objeto del dominio que se identifica de forma única y puede cambiar su estado a lo largo del tiempo sin dejar de ser el mismo.

- **Identidad única**

Tendrá un ID que lo distinguirá de los demás.

- **Mutable**

Su estado puede cambiar a lo largo de su ciclo de vida.

- **Coherencia y lógica de negocio**

Encapsula comportamiento y reglas de negocio.



Elementos de DDD: Agregado raíz

El agregado raíz es la entidad principal del agregado y el único punto de acceso al mismo desde el exterior.

- **Identidad**

Da identidad al agregado completo.

- **Punto de entrada**

Expone los comportamientos permitidos.

- **Garantía de invariantes**

Es responsable de que se cumplan las reglas del dominio.



¿Y si un agregado necesita comunicarse con otro agregado?

Un agregado no debería depender directamente de otros agregados para aplicar sus reglas.

- **Publica un evento**

El agregado publica un evento de dominio.

- **Reacción**

Un componente o contexto reacciona al evento y ejecuta la lógica correspondiente.

- **Independencia**

Cada contexto mantiene su independencia.



¿Y si un agregado depende de otro agregado?

Un agregado no contendrá agregados ni elementos de otros contextos.

- **Guarda el identificador**

Almacena el identificador del elemento de otro contexto.

- **Eventos**

La comunicación se realiza mediante eventos.



Vamos ahora con un ejemplo...

Queremos modelar una aplicación que gestione partidas de un juego. En cada partida participan varios jugadores y cada uno tiene una puntuación asociada a esa partida.

Se deben cumplir las siguientes reglas:

- Una partida no puede comenzar sin jugadores.
- Un jugador no puede aparecer dos veces en la misma partida.
- No se pueden modificar las puntuaciones cuando la partida ha terminado.



Así quedaría el dominio

AR	Partida
- id: UUID - estado: Creada Iniciada Finalizada - jugadores: Lista de JugadorEnPartida	
+ NO iniciar sin jugadores + NO jugadores duplicados + NO puntuar sin finalizar	

A	JugadorEnPartida
- id: UUID - nombre: NombreJugador - puntuacion: Puntuación	

VO	NombreJugador
- valor	
+ NO vacío	

VO	Puntuación
- valor	
+ NO negativo	



