DER

1) **Forma de notificación**: decidimos crear un atributo varchar en Persona que indique qué forma de notificación eligió la persona. Esto lo hicimos porque una forma de notificación es stateless y la otra statefull. Los atributos de la forma statefull los persistimos en dos tablas que referencian a persona, ya que existe una relacion one-to-many de persona hacia estas dos tablas.
   Para mapear la forma de notificación definimos un converter de String a FormaNotificación. Para esto tuvimos que transformar la clase SinApuros a stateless, de forma que el converter pueda devolver una única clase SinApuros independientemente de la Persona que se esté mapeando.

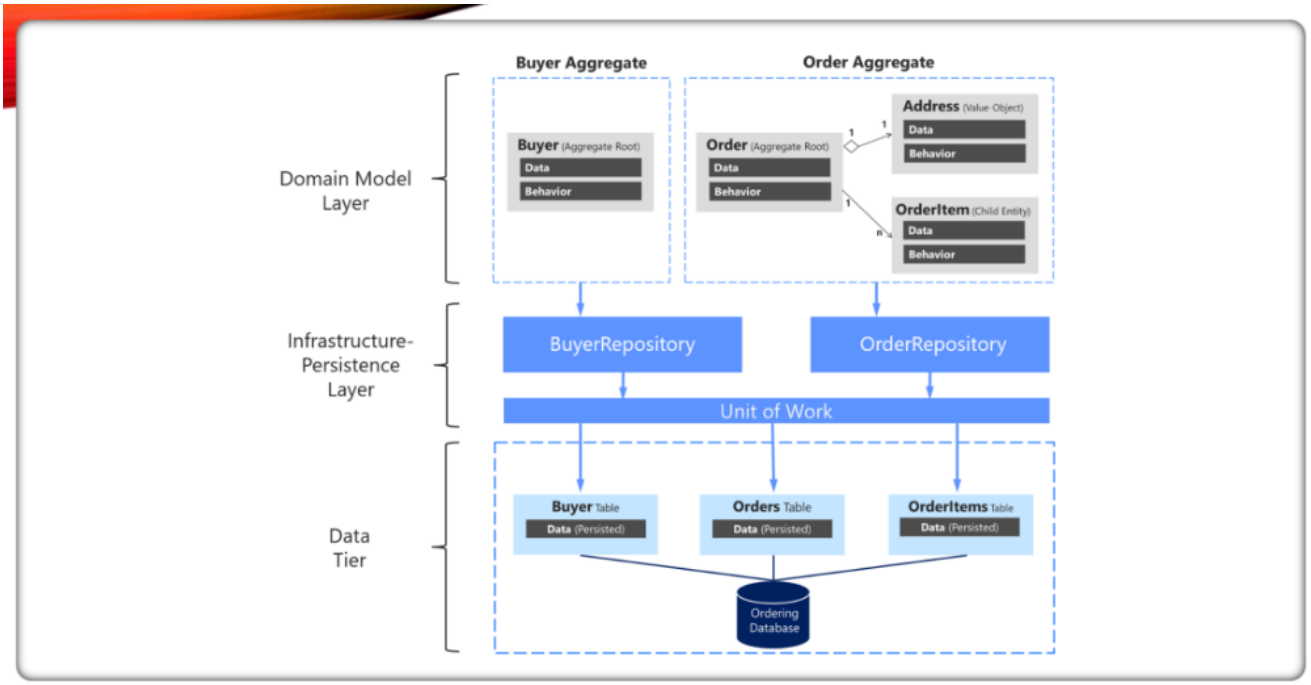
2) **ResultadoRanking**: en el código tenemos que los valores de los resultados de un ranking se guardan como un par ‘sujeto - valor’ donde sujeto puede ser o una Entidad o un Incidente, para poder lograr esto definimos 2 clases ValorRanking que implementan una interfaz con un método getSujeto(). Esta parte del dominio la decidimos persistir mediante una única tabla ResultadoRanking, de la cual existirá una fila por cada posición de cada ranking generado, las cuales contiene una referencia a Entidad y otra a Incidente, siendo siempre alguna de las dos null.

3) **Notificador:** para persistir qué notificador eligió la persona decidimos implementar una estrategia similar a la forma de notificación. En persona se guarda un campo varchar con el notificador elegido y, mediante un converter, se mapea de una String a una instancia de una clase que extienden el Notificador. Estas clases son todas Stateless, lo que permite que funcione el converter, ya que toma como entrada únicamente un string que referencia a un Notificador y no toma ningún dato específico de la Persona que se está mapeando.

REPOSITORIO

Al principio pensamos crear una clase repositorio por cada entidad que teníamos, pero con el tiempo nos dimos cuenta que se podría generalizarlos en una sola clase. Entonces decidimos crear una clase Repositorio junto a un FactoryRepositorio que, mediante generics, pueda instanciar repositorios de cualquier entidad.

La clase Repositorio delega todos sus métodos a un Dao, el cual implementamos como una interfaz en una suerte de Adapter para desacoplar la clase Dao de Hibernate. Entonces definimos la clase DaoHibernate que implementa Dao y define todos los métodos necesarios para poder realizar las operaciones de CRUD en la base de datos.


Buyer Aggregate
Order Aggregate
Domain Model Layer
Buyer (Aggregate Root)
Data
Behavior
Order (Aggregate Root)
Data
Behavior
Address (Value-Object)
Data
Behavior
OrderItem (Child Entity)
Data
Behavior
1
1
1
n
Infrastructure-Persistence Layer
BuyerRepository
OrderRepository
Unit of Work
Data Tier
Buyer Table
Data (Persisted)
Orders Table
Data (Persisted)
OrderItems Table
Data (Persisted)
Ordering Database