Patró repositori i DDD

Albert Clapés (aclapes@ub.edu)

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona (UB)

Projecte Integrat de Software (2024/25)

Objectius

Els **objectius** són:

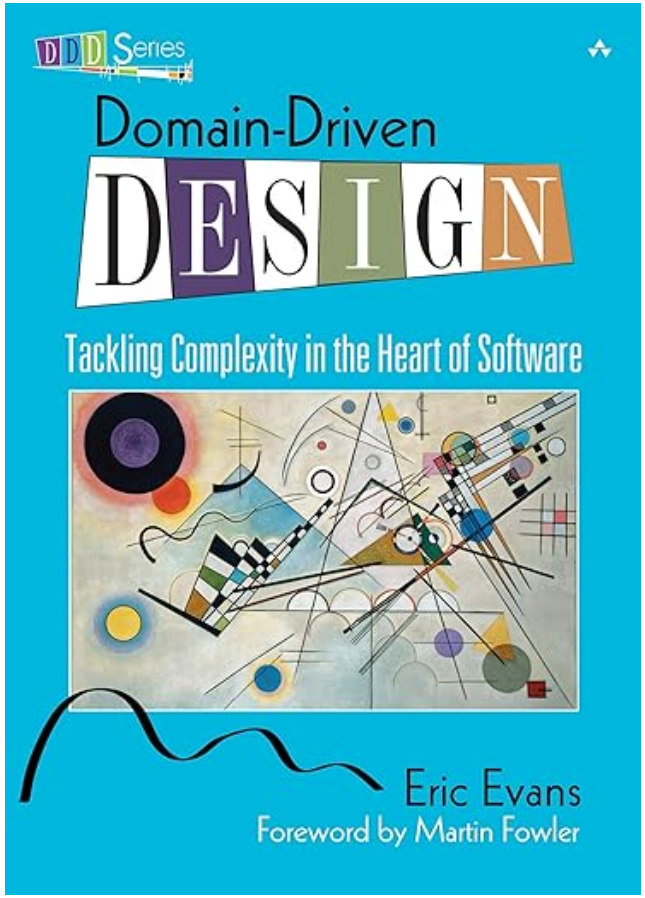
1. Introduir els conceptes bàsics de *Domain-driven Design* (DDD) per poder entendre el *patró repositori*.
2. Entendre el *patró repositori* i les seves variants per poder-lo aplicar en el projecte.

Context: *Domain-driven Design* (DDD)

El ***patró repositori*** té el seu origen en el DDD.

DDD és un enfocament per al disseny de programari que se centra en reflectir estrictament els requisits, models i regles del negoci en el codi.

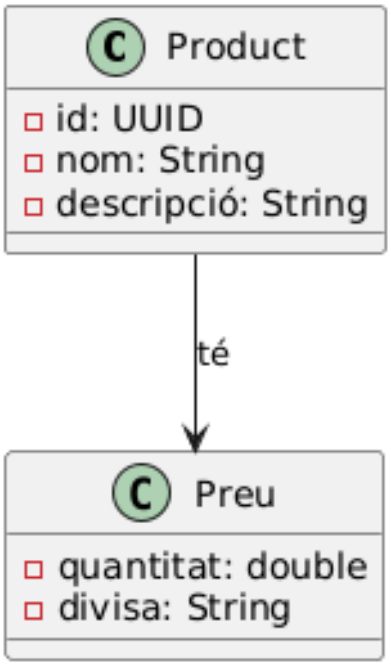
Defineix patrons **estratègics** (p. ex. *llenguatge omnipresent* o *contexts delimitats*) i **tàctics** (p. ex. *patró repositori* o *agregats*).



Abans dels agregats: entitats i objectes de valor

DDD distingeix els objectes del domini en:

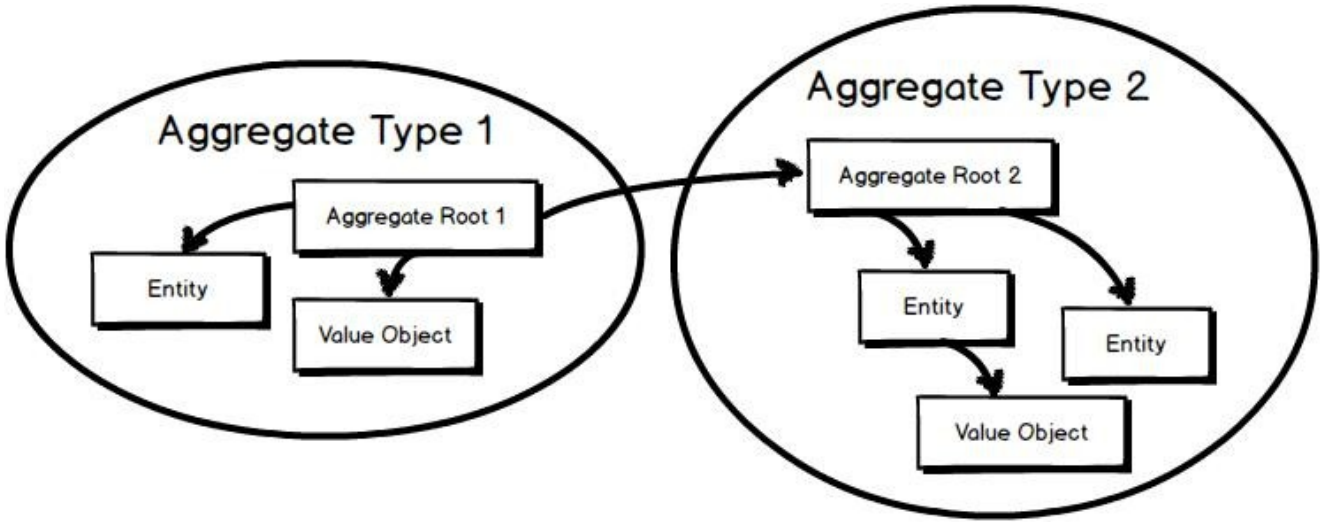
- **Entitats**. Objectes amb identitat, que s'identifiquen amb un atribut identificador únic i immutable.
  Exemple: un `Product` amb un `UUID`[a].
- **Objectes de valor**. Objectes sense identitat única, que s'identifiquen pel conjunt de valors de tots els seus atributs.
  Exemple: un `Preu` amb `quanitat` i `divisa`.



[a] Un *Universally Unique Identifier* és una etiqueta de 128-bit que serveix per identificar objectes. El número de combinacions és tan gran que, a la pràctica, resulten únics.

Agregats

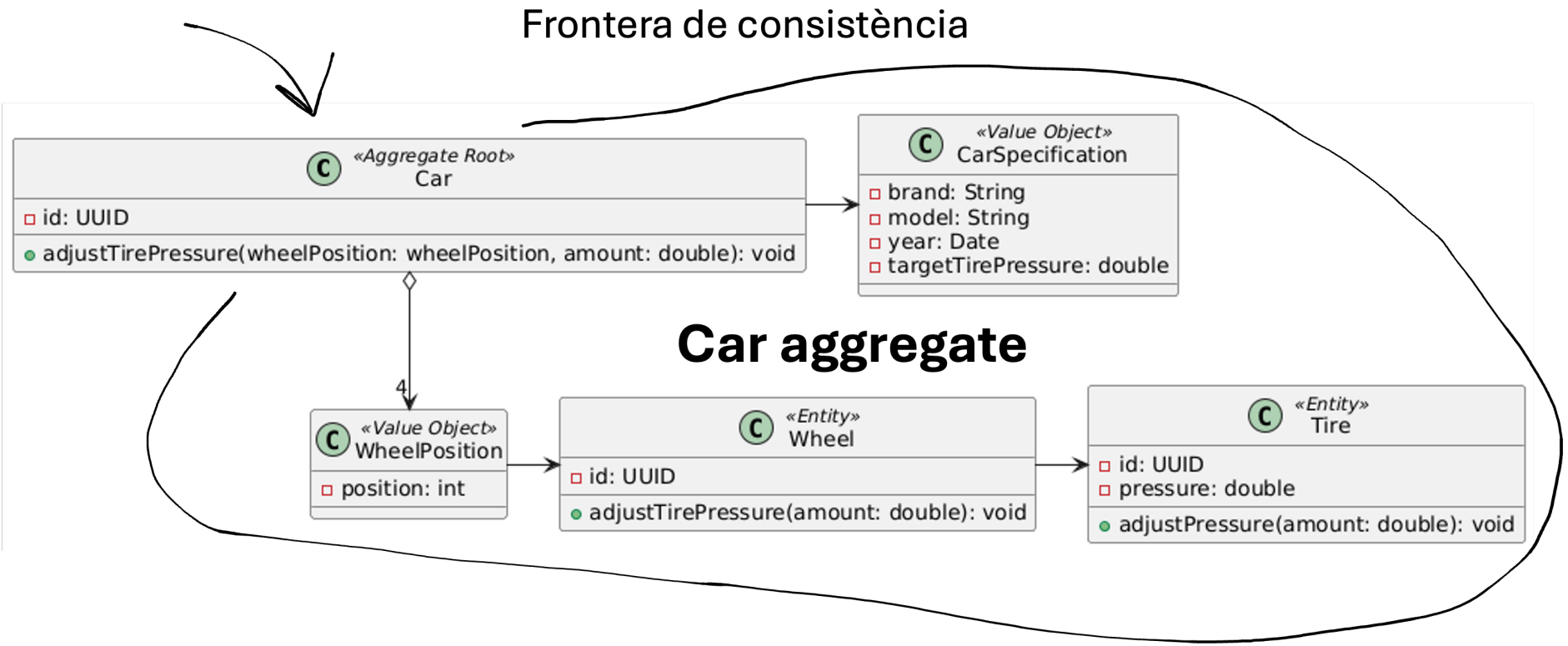
Un ***agregat*** és un conjunt d'objectes associats que tractem com una unitat per a l'efecte dels canvis de dades, establint una *frontera de consistència*.



L'***arrel de l'agregat*** és l'*entitat* que fa de punt d'entrada, exposant el comportament de la unitat i garantint la consistència de l'estat dels objectes en l'*agregat* i el seguiment de les regles de negoci.

Exemple d'agregat: Car

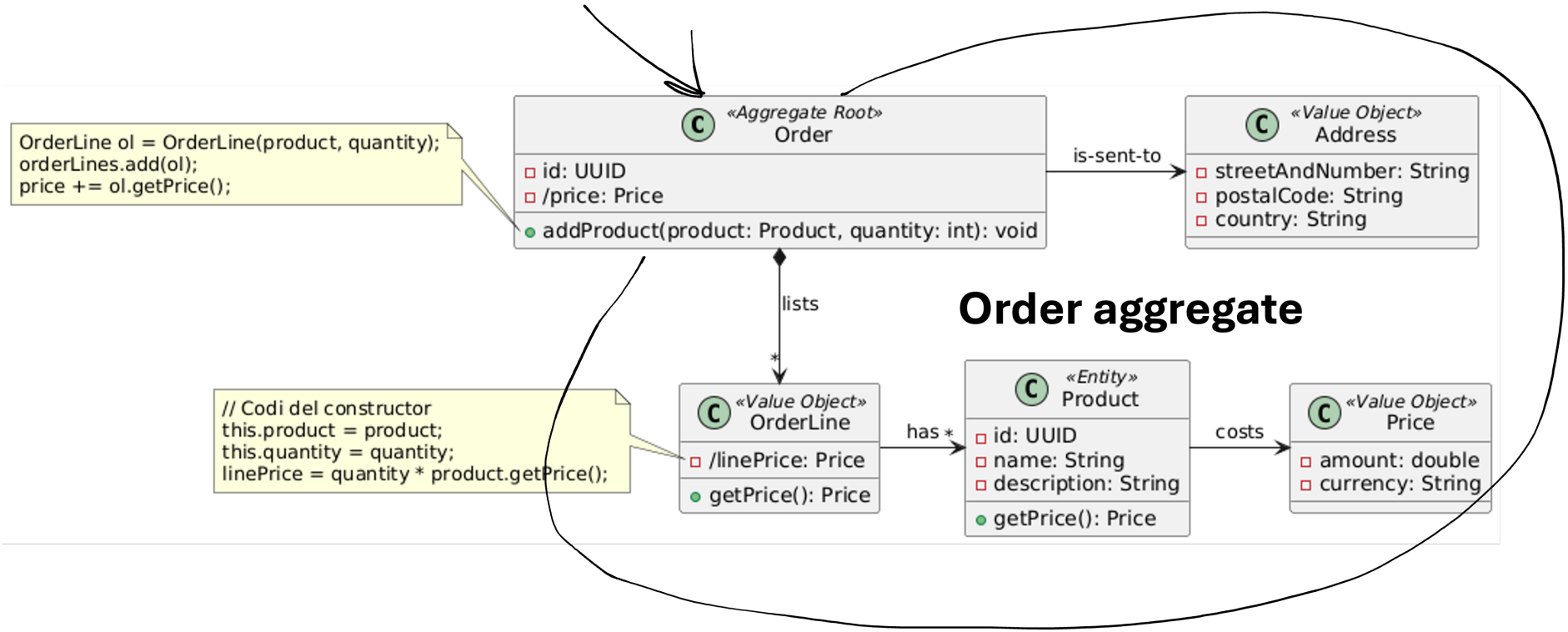
Un exemple.



Què passa si exposem un `Tire` i algú li ajusta la pressió sense que `Car` se n'adoni?

Exemple d'agregat: Order

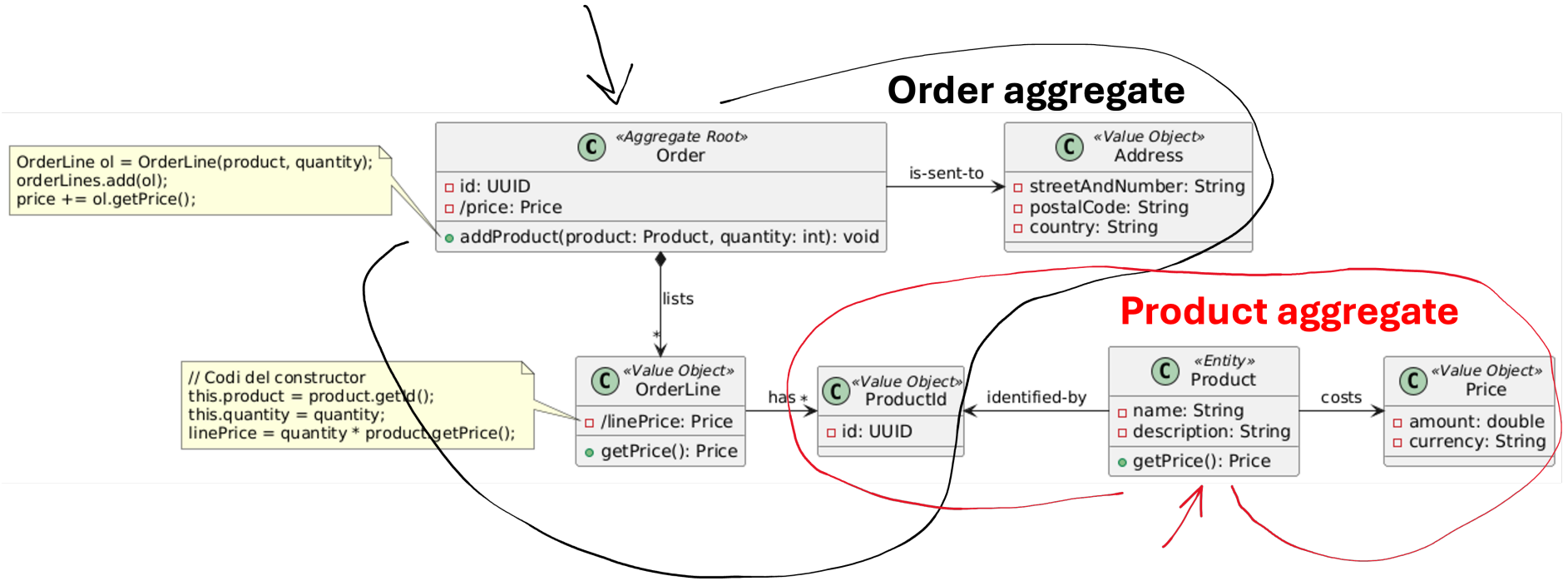
Un altre exemple.



I si, per aquest domini, tingués sentit poder accedir des de fora de l'agregat als `Product`?

Context: exemple d'agregat (Order)

Si volem accés global als `Product`, necessitem que tingui el seu propi agregat.



Per què hem creat `ProductId` i què passaria si l'agregat d'`Order` referenciés (atribut) als `Product`?

Agregats

Les regles per a la definició d'***agregats***:

- Han de tenir una *entitat arrel* per encapsular-ne el comportament i mantenir-ne la consistència.
- Poden incloure altres *entitats* i *objectes de valor*.
- Diferents *agregats* poden compartir objectes de valor, però no entitats.
- Cap dels seus elements pot mantenir una referència (atribut) a una *entitat* (arrel o no) d'un altre *agregat*. Només, i de manera transitòria (no guardar-la en cap atribut), fer ús de la referència a un altre *agregat* (és a dir, a la seva *entiat arrel*).
- ...

Agregats (*cont.*)

- ...
- Si una *entitat no-arrel* d'un *agregat*, ha de ser accessible des de fora de l'*agregat*, aquesta esdevindrà un nou *agregat*. I, per mantenir l'enllaç, utilitzarem un *objecte de valor* per representar la identitat de l'altre *agregat* (p. ex. `ProductId`).
- Fer-los el més petit possible sempre que la *frontera de consistència* tingui sentit.
- Si complim aquestes regles, són candidats ideals per esdevenir ***unitats de persistència***.

Serveis de domini

Els ***serveis de domini*** són objectes del domini que:

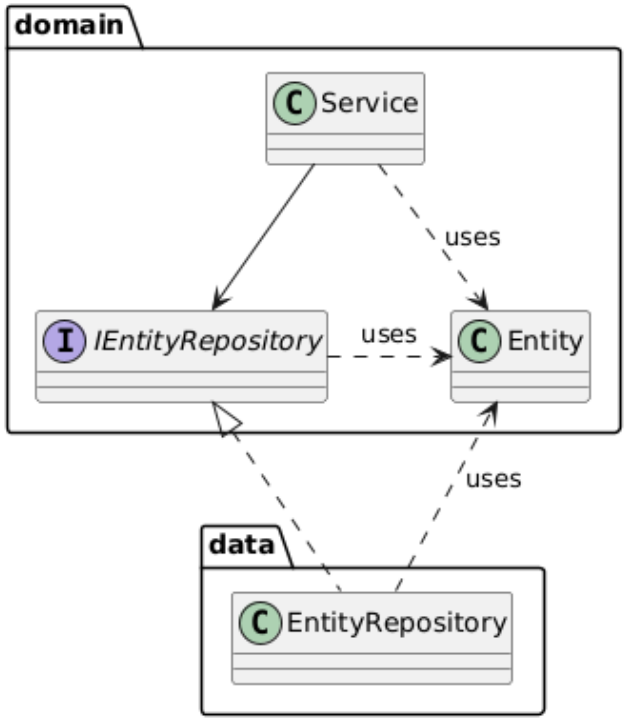
- Implementen els comportaments que no tenen cabuda* en cap *agregat*.
- S'encarreguen de fer persistir els *agregats*.
- No tenen estat.

(*) Quan un procés o transformació significatiu en el domini no és una "responsabilitat natural" d'una *entitat* o *objecte de valor*.

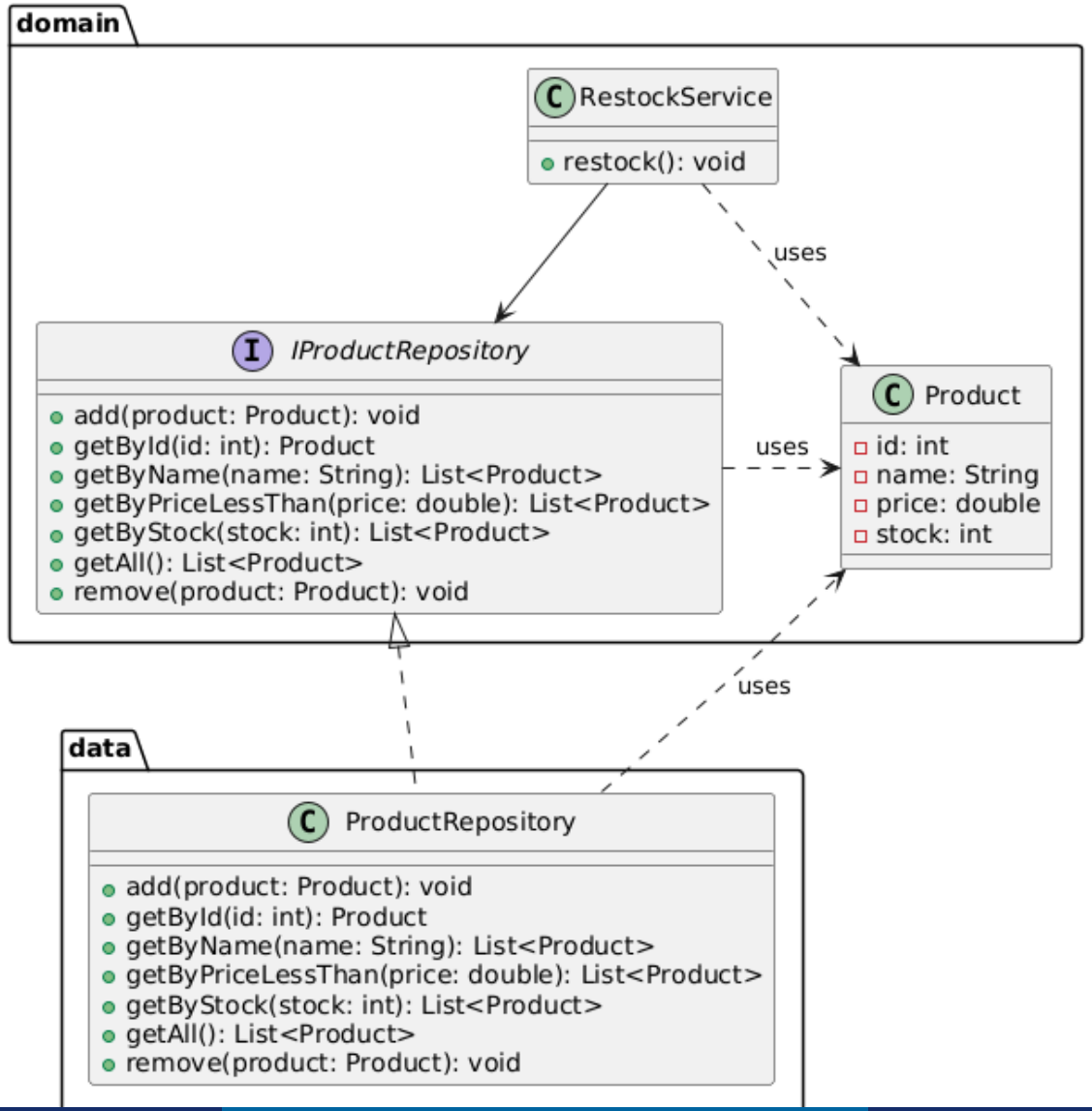
Patró repositori

El ***patró repositori*** és un patró de disseny estructural que defineix una abstracció per aïllar la lògica de negoci (capa del domini) de la lògica de persistència (capa de dades).

En termes de DDD, els *repositoris* se n'encarreguen de la persistència dels *agregats* i són invocats per mitjà de la seva l'abstracció (interfície) per part dels *serveis de domini*.



Exemple


domain
RestockService
restock(): void
uses
IProductRepository
add(product: Product): void
getById(id: int): Product
getByName(name: String): List<Product>
getByPriceLessThan(price: double): List<Product>
getByStock(stock: int): List<Product>
getAll(): List<Product>
remove(product: Product): void
uses
Product
id: int
name: String
price: double
stock: int
uses
data
ProductRepository
add(product: Product): void
getById(id: int): Product
getByName(name: String): List<Product>
getByPriceLessThan(price: double): List<Product>
getByStock(stock: int): List<Product>
getAll(): List<Product>
remove(product: Product): void

Variants del *patró repositori*

Existeixen dues variants del patró:

- Específic
- Genèric

Patró *repositori específic*

El ***patró repositori especific*** defineix una abstracció (interfície) per cada *entitat* (arrel d'un *agregat*) amb mètodes d'obtenció pels diferents criteris segons els atributs de l'*entitat*.

```
public interface IEntityRepository {
    void add(Entity entity);
    Entity getById(EntityId id);
    // ... altres getBy ...
    Iterable<Entity> getAll();
    void remove(Entity entity);
}
```

Patró *repositori genèric*

El ***patró repositori genèric*** defineix una abstracció comuna que qualsevol repositori ha d'implementar.

```
public interface IRepository<E> {
    void add(E e);
    Iterable<E> getBy(Criterion<E> criterion);
    Iterable<E> getAll();
    void remove(E e);
}
```

El mètode `getBy(...)` es basa en un criteri genèric (l'abstracció `Criterion<T>`) que el codi client (*servei*) concretarà:

```
public interface Criterion<E> {
    boolean isSatisfiedBy(E e);
}
```

Patró *repositori genèric* (*cont.*)

Exemple de criteri concret

```
public class StockLowerThanCriterion implements
↪  Criterion<Product> {
    int amount;

    StockLowerThanCriterion(int amount) {
        this.amount = amount;
    }

    public boolean isSatisfiedBy(Product product) {
        return product.getStock() < amount;
    }
}
```

Patró *repositori genèric* (*cont.*)

Exemple d'implementació *repositori genèric*

```
public class ProductInMemoryRepository implements
↪  IRepository<Product> {
    ...

    public Iterable<Product> getBy(Criterion<Product> criterion) {
        List<Product> matchingProducts = new ArrayList<>();

        for (Product p : this.getAll()) {
            if (criterion.isSatisfiedBy(p) {
                matchingProducts.add(p);
            }
        }

        return matchingProducts;
    }

    ...
}
```

Patró *repositori genèric* (*cont.*)

Exemple de codi client

```
IRepository<Product> productRepository = new
↪  ProductRepository();
...
Criterion<Product> criterion = new
↪  StockLowerThanCriterion(10);
Iterable<Product> products =
↪  productRepository.getBy(criterion);

// Buy more stock from distributors
distributorService.purchase(products, 30);
```

Patró *repositori específic*

Tornem al *repositori específic* per veure la diferència del codi client.

Exemple de codi client

```
IProductRepository productRepository = new
↪  ProductRepository();
...
Iterable<Product> products =
↪  productRepository.getByStockLowerThan(10);

// Buy more stock from distributors
distributorService.purchase(products, 30);
```

Instanciació del *repositori*

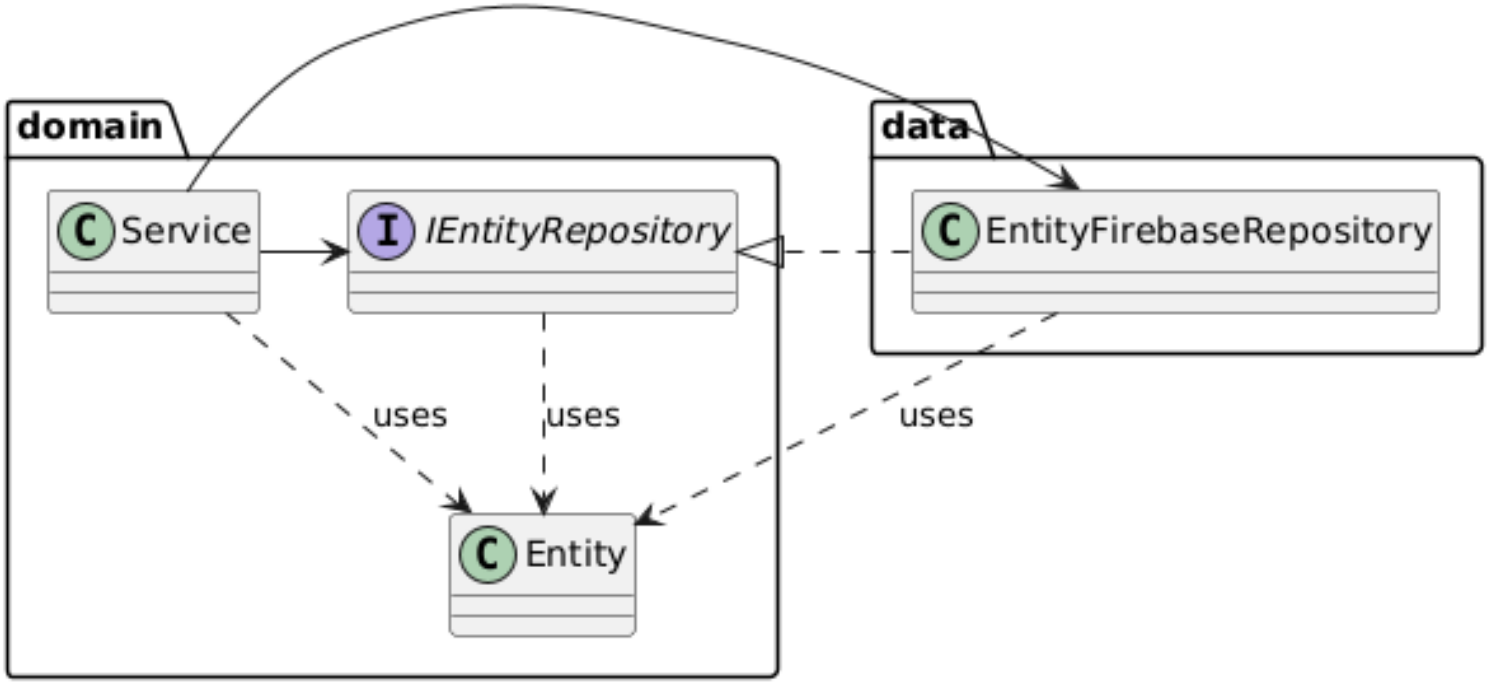
Si instanciem el *repositori* en el *servei*:

```
public class RestockService {
    private IProductRepository productRepository;

    public RestockService() {
        productRepository = new
        ↪  ProductFirebaseRepository();
    }

    public void execute(...) {
        ...
    }
}
```

Instanciació del *repositori* (*cont.*)



Injecció d'abstracció del repositori

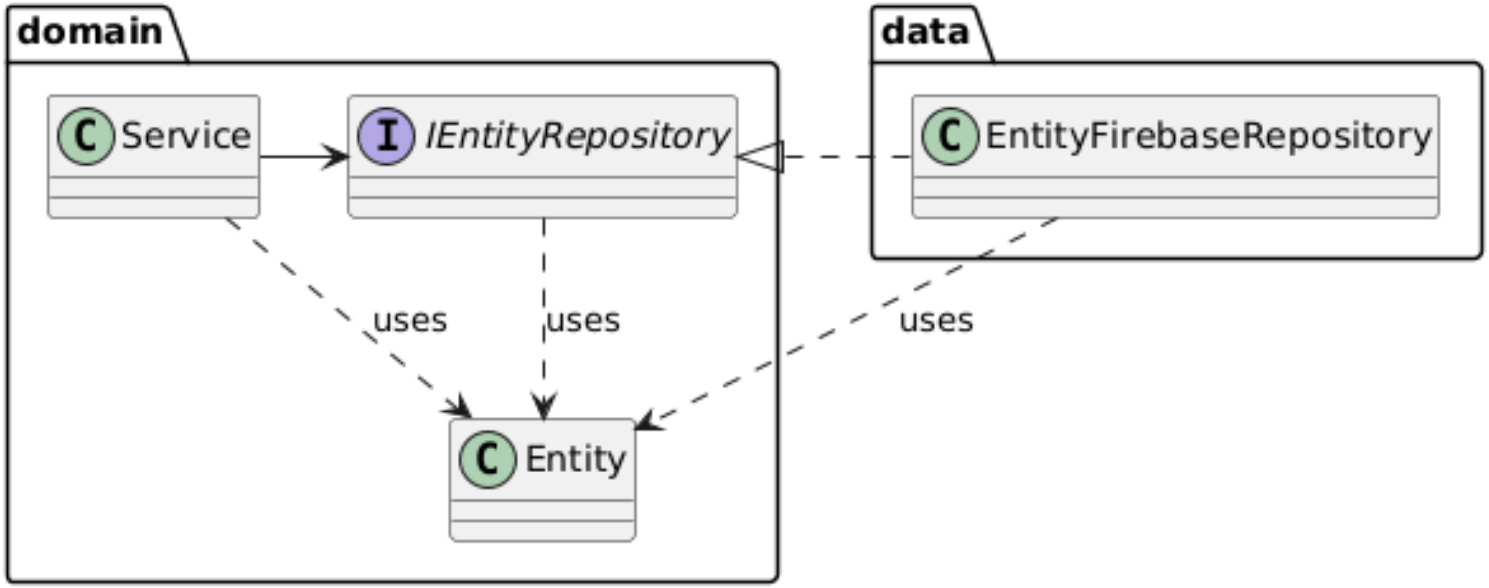
Si injectem la abstracció (interfície) del *repositori* pel constructor del *servei*:

```
public class RestockService {
    private IProductRepository productRepository;

    public RestockService(IProductRepository
     ↪  productRepository) {
        productRepository = productRepository;
    }

    public void execute(...) {
        ...
    }
}
```

Injecció d'abstracció del repositori (*cont.*)



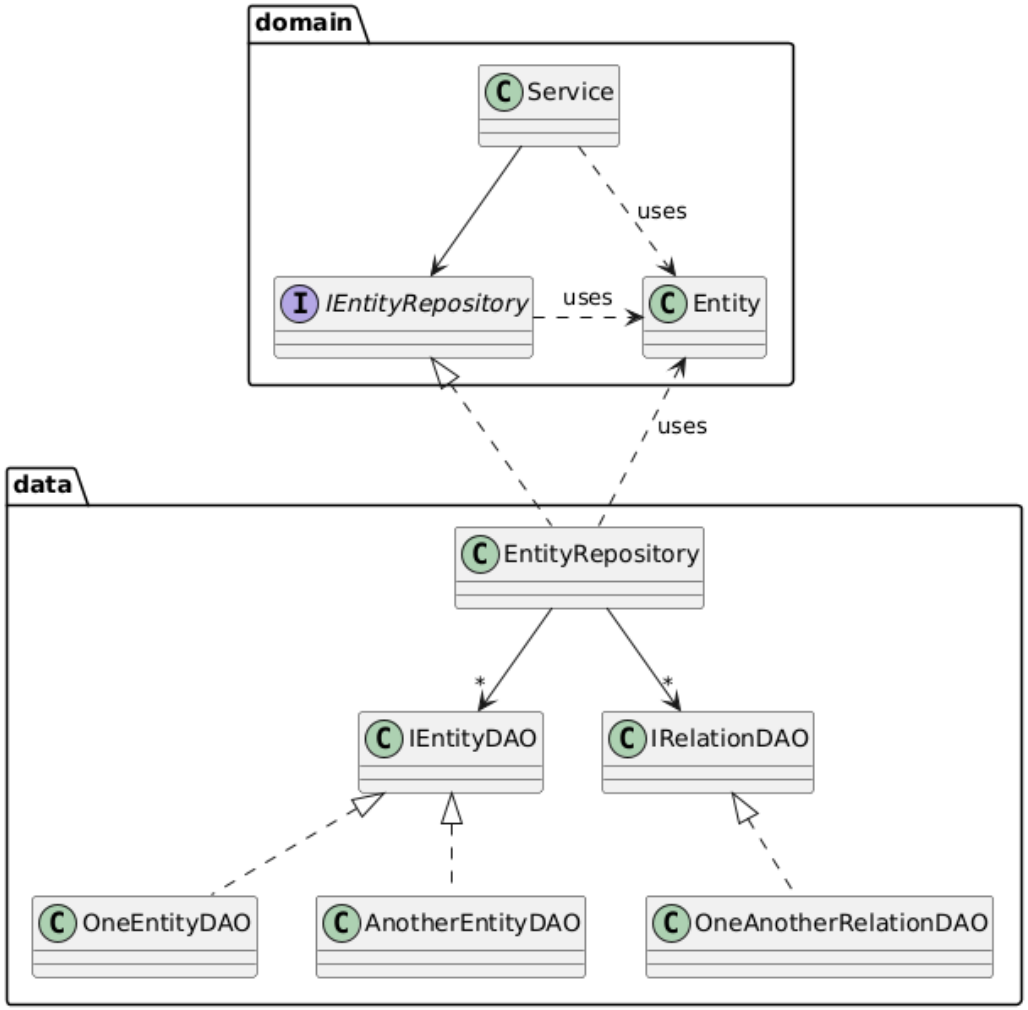
Repositori vs. DAO

Tot i la semblança, existeixen diferències tècniques (o a vegades filosòfiques):

Repositori	**DAO**
Accessible des de la capa de domini	No accessible des de la capa de domini (o no es discuteix en el patró)
Opera, directament, amb unitats de persistència (agregats)	No opera amb unitats de persistència
Més allunyat de l'estructura de la base de dades	Més proper a la base de dades (un DAO per entitat i un DAO per relació)
Més flexible en termes de comportament (`getById(...)`)	Estrictament CRUD

Repositori vs. DAO (cont.)

No són mutuament excloents. Es poden combinar.



Bibliografia

- Eric Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley, 2003.
- Vaughn Vernon, *Implementing Domain-Driven Design*, Addison-Wesley, 2013.
- Vaughn Vernon, *Domain-Driven Design Distilled*, Addison-Wesley, 2016.
- Martin Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2002.