Repository

Repository

- abstrakcja źródła danych
  - decoupling źródeł danych – może być ich wiele
  - mogą wstrzykiwać zależności do odtwarzanych klas
  - zapewniają blokowanie optymistyczne
- zarządza utrwalaniem obiektu
  - pobiera po ID lub kryteriach biznesowych
  - nie należy używać jak wyszukiwarki

Repository

```
public interface IOfferRepository : IRepository<Offer>
{
   Offer GetById(Offer.DomainId id);
   IEnumerable<Offer> GetAll();
   void Add(Offer offer);
}
```

Repository

- Może być wstrzykiwane i do Application i Domain Services - tak!
- Repozytorium jest po to, by móc kiedyś wymienić DB/DAL - jest to tylko efekt uboczny. Zresztą jedno repozytrium może mieć różne DAL per metoda (np. Query wali ADO.NET SQLem a Command używają EF itd itp
- Repozytorium ułatwia unit testy - tak, ale nie unit testy DAL tylko logiki biznesowej (domeny)
- Generyczne repozytorium - nie ma potrzeby (YAGNI), dodawaj tylko to co potrzebne
  - Repozytorium zwracające **`IQueryable`**:

```
public interface IRepository<TEntity>
{
IQueryable<TEntity> Query();
// [...]
}
```

  - Exposing **`IQueryable<T>`** jako antypattern - wycieka wiedza

Repository

- Może być wstrzykiwane i do Application i Domain Services - tak!
- Repozytorium jest po to, by móc kiedyś wymienić DB/DAL - jest to tylko efekt uboczny. Zresztą jedno repozytrium może mieć różne DAL per metoda (np. Query wali ADO.NET SQLem a Command używają EF itd itp
- Repozytorium ułatwia unit testy - tak, ale nie unit testy DAL tylko logiki biznesowej (domeny)
- Generyczne repozytorium - nie ma potrzeby (YAGNI), dodawaj tylko to co potrzebne
  - Repozytorium zwracające **`IQueryable`**:

```
public interface IRepository<TEntity>
{
IQueryable<TEntity> Query();
// [...]
}
```

  - Exposing **`IQueryable<T>`** jako antypattern - wycieka wiedza

"Generyczne repozytorium plus EF to rak. Szczególnie wystawiając `IQueryable`"

Repository

Może być potrzeb wielu podejść w zależności od punktu wejścia:

Repository

Może być potrzeb wielu podejść w zależności od punktu wejścia:

- UI/WebAPI - może mu wystarczyć zwykły CRUD i bezpośredni dostęp (ADO.NET, Dapper)

Repository

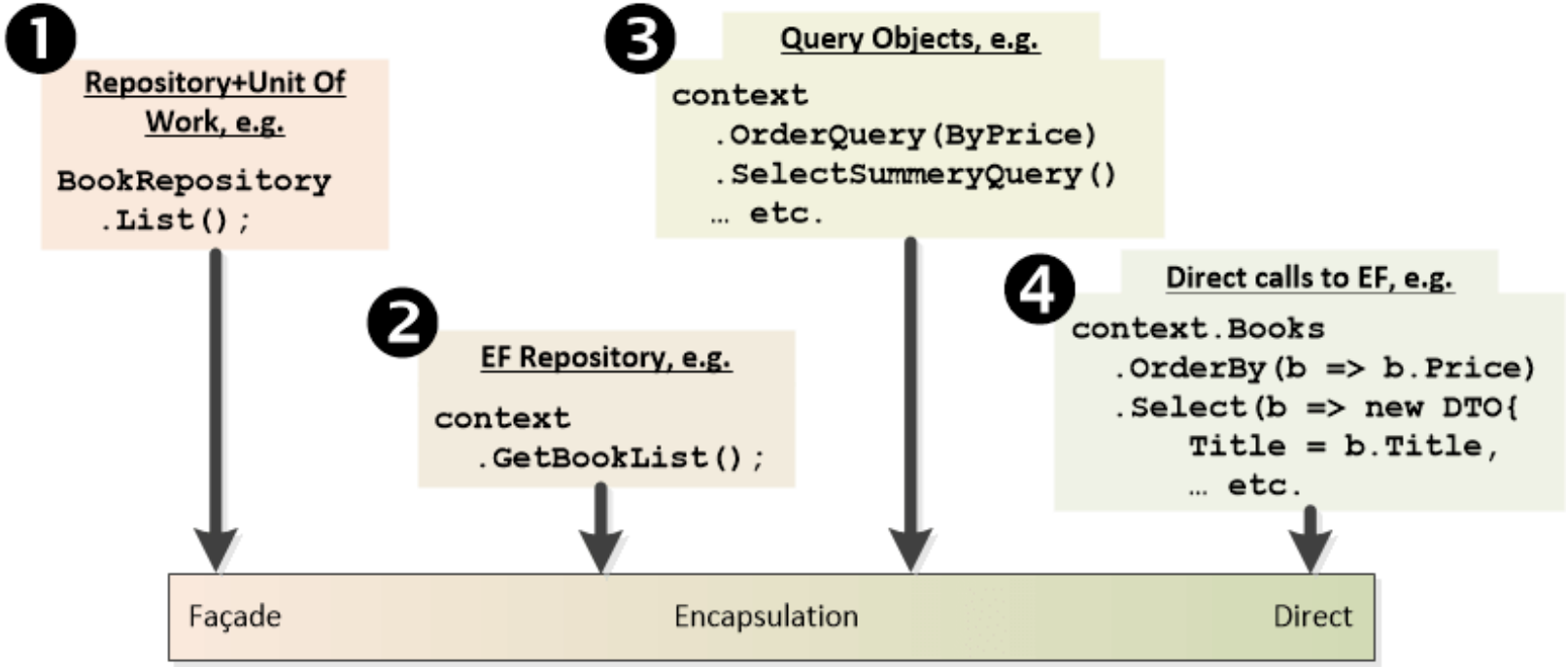
Może być potrzeb wielu podejść w zależności od punktu wejścia:

- UI/WebAPI - może mu wystarczyć zwykły CRUD i bezpośredni dostęp (ADO.NET, Dapper)
- BL - tutaj cięższe armaty (EF)

Repository

Może być potrzeb wielu podejść w zależności od punktu wejścia:

- UI/WebAPI - może mu wystarczyć zwykły CRUD i bezpośredni dostęp (ADO.NET, Dapper)
- BL - tutaj cięższe armaty (EF)



Repository - typowe błędy

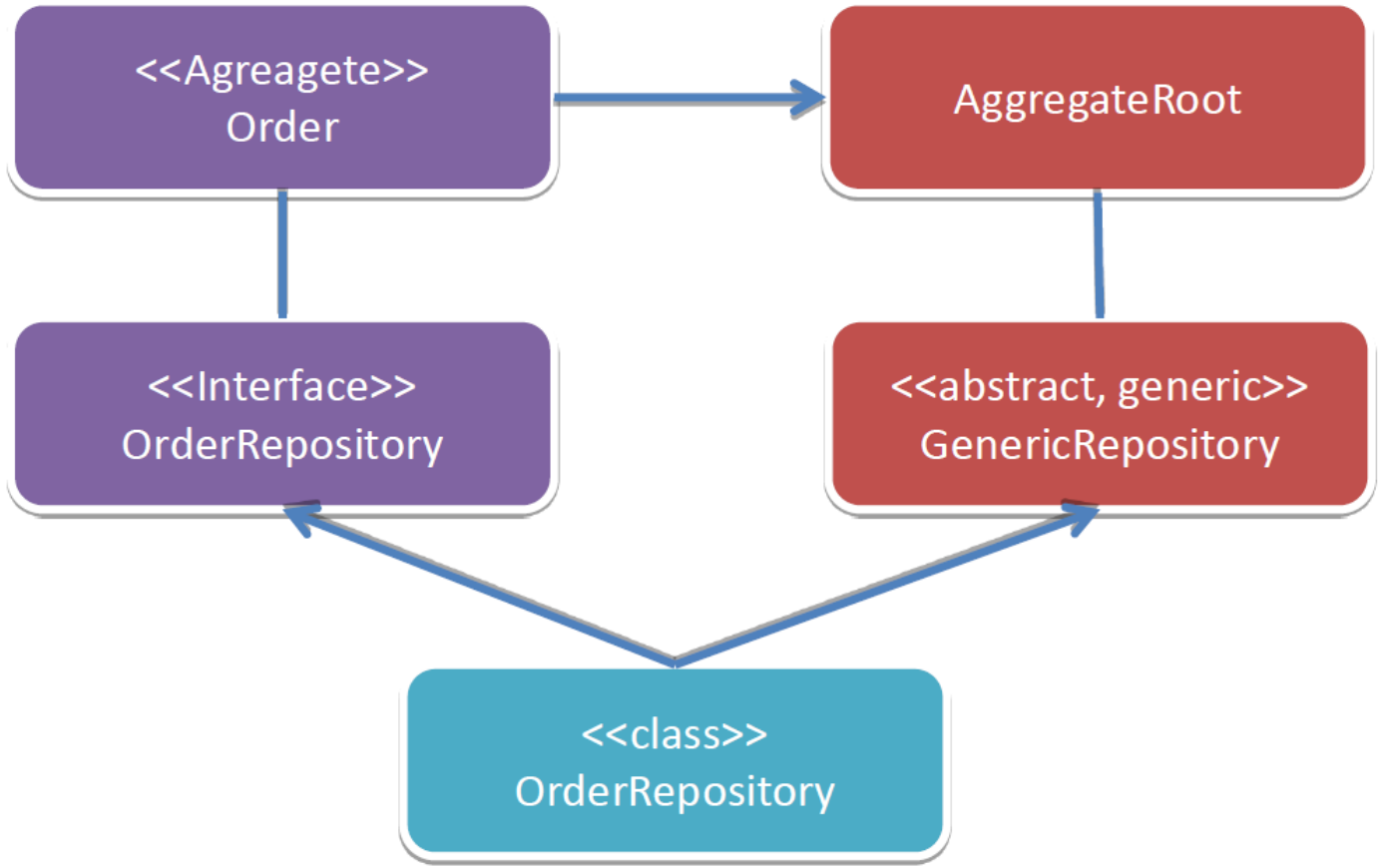
- IQuerable
- Interfejs generyczny do repository

```
public interface IRepository<TEntity>
{
int Add(TEntity obj)
Update(TEntity obj)
Remove(int id)
TEntity Get(int id)
IQueryable<TEntity> Query();
}
```

- Logika – **IPrincipal**

```
public class IRepository
{
int Add(TEntity obj)
{
   ...
   cod.Parameters.Add("UserName”, Thread.CurrentPrincipal.Identity.Name)
   ...
```

Repository - generic repository



Repository - generic repository

```
public class GenericRepository<TEntity> : IGenericRepository<TEntity>
{
   private readonly IDependencyInjector _dependencyInjector;
   private readonly ISession _session;
   public GenericRepository(ISession session,
      IDependencyInjector dependencyInjector)
   {
      _session = session;
     _dependencyInjector = dependencyInjector;
   }
   public TEntity Load(int id)
   {
      var result = _session.Get<TEntity>(id);
      _dependencyInjector.InjectDependencies(result);
      return result;
   }
   public void Save(TEntity aggregateRoot)
   {
      _session.SaveOrUpdate(aggregateRoot);
   }
   public void Delete(int id)
   {
      _session.Delete(_session.Get<TEntity>(id));
   }
}
```