

Tehnici Avansate de Programare

Unit of Work pattern, Dependency Injection &
Test-Driven Development

Petru Rebeja, Marius Apetrii

9 Aprilie 2020

Facultatea de Matematică

Universitatea Alexandru Ioan Cuza, Iași

Introdurre

Recapitulare

- **LINQ** — O mulțime de funcții ce permit interogarea uniformă a colecțiilor de date din diverse surse.
- **Entity Framework** — O platformă care permite interogarea și manipularea datelor folosind paradigma orientat-obiect.
- **Repository Pattern** — Șablon de proiectare ce decuplează logica aplicației de accesul la date.

Agenda

- Unit of Work pattern
- Dependency Injection
- Test-Driven Development

Unit of Work **pattern**

```
public interface IRepository<TEntity>
{
    IQueryable<TEntity> Query();
    void Insert(TEntity entity);
    void Update(TEntity entity);
    void Delete(TEntity entityId);
    void SaveChanges();
}
```

- Folosim o instanță de `repository` pentru a efectua operații CRUD asupra uneia sau a mai multor entități de același tip.
- Toate modificările sunt păstrate în memorie.
- Metoda `SaveChanges()` este apelată pentru a salva modificările în baza de date.

Vrem să introducem în baza de date un client nou și datele de contact ale acestuia:

```
customerRepository.Insert(new Customer{...});  
customerRepository.SaveChanges();
```

```
addressRepository.Insert(new CustomerAddress{...});  
addressRepository.SaveChanges();
```


Vrem să introducem în baza de date un client nou și datele de contact ale acestuia:

```
customerRepository.Insert(new Customer{...});  
customerRepository.SaveChanges();
```

```
addressRepository.Insert(new CustomerAddress{...});  
addressRepository.SaveChanges();
```

Ce se întâmplă dacă `addressRepository.SaveChanges()` aruncă o excepție?

Ce se întâmplă dacă `addressRepository.SaveChanges()` aruncă o excepție?

- Clientul este salvat în baza de date dar adresa nu.
- Baza de date rămâne într-o stare neconsistentă.
- Pentru a reveni la consistență clientul fără adresă trebuie șters.

- Astfel de scenarii se regăsesc des în cerințele aplicațiilor.
- Atunci când lucrăm cu mai mult de două entități asigurarea consistenței datelor devine un mecanism foarte complex.

Pentru a evita astfel de probleme folosim Unit of Work.

Unit of Work

Unit of Work ține evidența tuturor modificărilor aferente bazei de date efectuate în timpul unei tranzacții logice din aplicație. La finalul tranzacției modificările sunt salvate în baza de date într-un mod care asigură atomicitatea și integritatea datelor.¹

¹<https://www.martinfowler.com/eaCatalog/unitOfWork.html>

Unit of Work Pattern

```
public interface IRepository<TEntity>
{
    IQueryable<TEntity> Query();
    void Insert(TEntity entity);
    void Update(TEntity entity);
    void Delete(TEntity entityId);
}
```

```
public interface IUnitOfWork
{
    void Commit();
}
```

- Metoda `SaveChanges()` devine redundantă și trebuie eliminată din interfața `IRepository`.

Exemplu

```
public class RegisterCustomerTransaction
{
    public RegisterCustomerTransaction(
        IRepository<Customer> customerRepository,
        IRepository<CustomerAddress> addressRepository,
        IUnitOfWork unitOfWork)
    {
        ...
    }

    public void Execute(Customer customer, CustomerAddress address)
    {
        customerRepository.Insert(customer);
        addressRepository.Insert(address);
        unitOfWork.Commit();
    }
}
```

Depencency Injection

Dependency Injection

Dependency Injection este o tehnică prin care un obiect îi furnizează alțiia dependențele de care acesta din urmă are nevoie.²

²https://en.wikipedia.org/wiki/Dependency_injection

Dependency Injection

Dependency Injection este un termen de 25 dolari pentru un concept de 5 cenți. [...] Dependency Injection înseamnă să-i dai unei instanțe a unui anumit obiect variabilele de care are nevoie.³

³[https:](https://www.jamesshore.com/Blog/Dependency-Injection-Demystified.html)

[//www.jamesshore.com/Blog/Dependency-Injection-Demystified.html](https://www.jamesshore.com/Blog/Dependency-Injection-Demystified.html)

- Separarea responsabilităților⁴:
 - **Injectorul** construiește dependențele și le injectează în client și
 - **Clientul** manipulează dependențele pentru obținerea unui anumit scop.
- Reducerea codului de umplură și a gradului de acuplare.

⁴https://en.wikipedia.org/wiki/Dependency_injection

- Flexibilitatea la schimbări fără necesitatea de recompilare — efectele execuției codului client pot fi modificate prin injectarea unei implementări diferite.
- Facilitează testarea în izolare.

- Grad de abstractizare mai mare care crește dificultatea codului.
- Induce o dependență față de o bibliotecă pentru `Dependency Injection`.
- Poate duce la o acuplare mai mare prin utilizarea defectuoasă.

Service Locator

Ideea de bază a unui Service Locator este să ai la dispoziție un obiect care știe cum să obțină, la cerere, instanțe ale tuturor serviciilor necesare unei aplicații⁵.

N.B.: O instanță de serviciu = o dependență.

⁵[https:](https://martinfowler.com/articles/injection.html#UsingAServiceLocator)

[//martinfowler.com/articles/injection.html#UsingAServiceLocator](https://martinfowler.com/articles/injection.html#UsingAServiceLocator)

Service Locator — exemplu

```
public class RegisterCustomerTransaction
{
    public RegisterCustomerTransaction()
    {
        customerRepository = ServiceLocator
            .GetInstance<IRepository<Customer>>();
        addressRepository = ServiceLocator
            .GetInstance<IRepository<CustomerAddress>>();
        unitOfWork = ServiceLocator.GetInstance<IUnitOfWork>();
    }
}
```

- Este un anti-șablon⁶.
- Induce un grad mare de acuplare.
- Ascunde dependențele.

⁶<https://blog.ploeh.dk/2010/02/03/ServiceLocatorisanAnti-Pattern/>

Test-Driven Development

Test-Driven Development⁷

Este un stil de dezvoltare a aplicațiilor constând în trecerea frecventă prin următoarele trei activități:

- Scrierea de cod
- Scrierea de teste unitare și
- Refactorizare.

⁷<https://www.agilealliance.org/glossary/tdd>

- Sciirea unui singur test unitar menit să descrie un anumit aspect al metodei,
- Execuția testului care eșuează deoarece aspectul încă nu este implementat,
- Scrierea cantității de cod necesare pentru a trece testul,
- Refactorizarea codului scris pentru a se încadra în diverse criterii (simplitate, lizibilitate etc.).

⁸<https://www.agilealliance.org/glossary/tdd>

Procesul se repetă de mai multe ori ducând la acumularea de teste unitare pentru fiecare aspect al aplicației.

- Reduce timpul petrecut pentru depanare,
- Reduce numărul defectelor din aplicație,
- Scade gradul de acuplare și crește coeziunea.

Dezavantaje

- Crește timpul necesar pentru dezvoltare,
- Crește efortul necesar întreținerii,
- Necesită o perioadă de adaptare.

- Metode care verifică un anumit aspect al modulului pe care îl testează.
- Sunt decorate cu atributul `[TestMethod]`.
- De obicei sunt într-o clasă decorată cu `[TestClass]`.

Un test este structurat conform șablonului AAA:

- **Arrange** — pregătirile necesare pentru a apela metoda testată: crearea de instanțe false, instanțierea clasei etc.
- **Act** — apelarea propriu-zisă a metodei testate și preluarea rezultatelor.
- **Assert** — verificarea rezultatelor și/sau comportamentului metodei testate.

Înceiere

Recapitulare

Unit of Work

Unit of Work

Este un șablon care ne permite să executăm toate modificările aferente bazei de date într-o singură tranzacție.

Unit of Work

Este un șablon care ne permite să executăm toate modificările aferente bazei de date într-o singură tranzacție.

Dependency Injection

Unit of Work

Este un șablon care ne permite să executăm toate modificările aferente bazei de date într-o singură tranzacție.

Dependency Injection

Este o modalitate de a-i da unei instanțe variabilele de care aceasta are nevoie separând astfel crearea de instanțe de utilizarea lor.

Unit of Work

Este un șablon care ne permite să executăm toate modificările aferente bazei de date într-o singură tranzacție.

Dependency Injection

Este o modalitate de a-i da unei instanțe variabilele de care aceasta are nevoie separând astfel crearea de instanțe de utilizarea lor.

Test-Driven Development

Unit of Work

Este un șablon care ne permite să executăm toate modificările aferente bazei de date într-o singură tranzacție.

Dependency Injection

Este o modalitate de a-i da unei instanțe variabilele de care aceasta are nevoie separând astfel crearea de instanțe de utilizarea lor.

Test-Driven Development

Este un stil de dezvoltare software în care mai întâi se scriu testele pentru un anumit aspect iar mai apoi implementarea propriu-zisă.

Vă mulțumesc!

Mulțumesc pentru atenție!