DISSENY DE LA CAPA DE PERSISTÈNCIA

Seminari T6 -2 – Desenvolupament de Programari en Visual Studio 2022

Enginyeria del Programari

DSIC-UPV Curs 2024-2025

Objectius

- Aprendre a dissenyar la capa de persistència aplicant patrons de disseny i principis de separació de capes

- Aplicar els coneixements de **Entity Framework** per a la implementació de la capa de persistència

Patrons de disseny per a la capa de persistència (accés a dades)

- En dissenyar la capa de persistència estem interessats en:
  - Abstraure els detalls del mecanisme d'implementació a les altres capes
  - Facilitar un hipotètic canvi en el mecanisme concret de persistència

- En classe de teoria hem vist dos patrons de disseny apropiats per a aqueixos objectius
  - Patró DAO
  - Patró Repository + UoW

Persistència i Capa de Accés a Dades

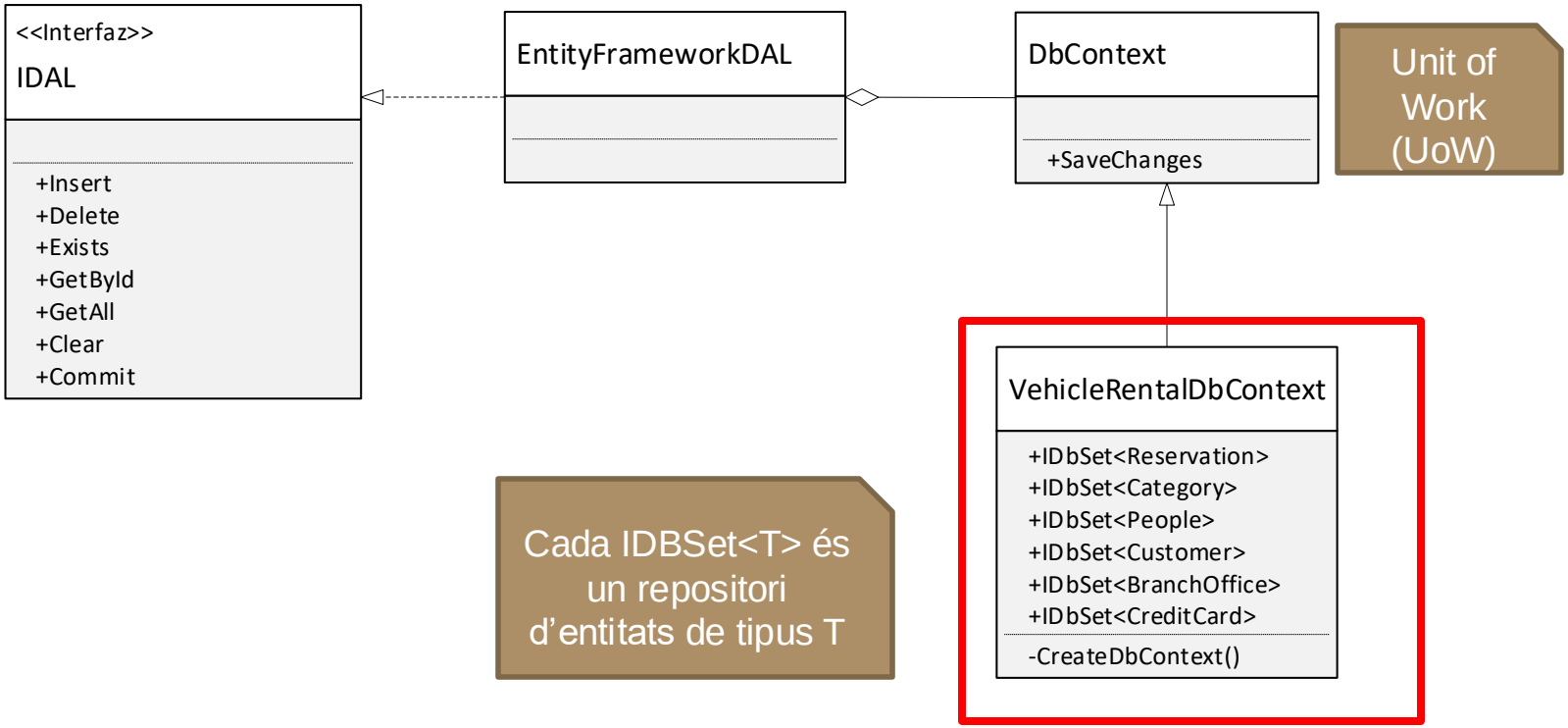
- En parlar de **persistència** ens referim a tota la infraestructura encarregada de persistir i/o recuperar dades en/des d'algun mecanisme d'emmagatzematge, com una base de dades, un arxiu o un servei Web.

- Cridarem **Capa d'Accés a Dades** (Data Access Layer, abreujat **DAL**) a la part de la persistència encarregada de donar servei altres capes (en la nostra arquitectura tancada de 3 capes donarà servei a la capa de negoci)

Disseny de la capa de persistència

- En les pràctiques de laboratori anem a desenvolupar una aplicació d'escriptori de tamany xicotet/mitja, simplificant l'arquitectura encara que utilitzant patrons reconeguts i seguint bones pràctiques.
  - Com a mecanisme concret de persistència gastarem un framework ORM, **Entity Framework**, el qual implementa els patrons **Repository + UoW** per a l'accés a dades.
  - Per a desacoplar la capa d'accés a dades del mecanisme d'implementació gastarem una interfície pública genèrica amb tots els mètodes necessaris, denominada **IDAL**

Disseny arquitectònic (capa de persistència)



Interfície IDAL

```
public interface IDAL
    {
      void Insert<T>(T entity) where T : class;
      void Delete<T>(T entity) where T : class;
      bool Exists<T>(IComparable id) where T : class;
      T GetById<T>(IComparable id) where T : class;
      IEnumerable<T> GetAll<T>() where T : class;
      void Clear<T>() where T : class;
      void Commit();
}
```

DAL amb Entity Framework

```
public class EntityFrameworkDAL : IDAL
{
  private readonly DbContext dbContext;

  public EntityFrameworkDAL(DbContext dbContext)
  {
    this.dbContext = dbContext;
  }

  public void Insert<T>(T entity) where T : class
  {
    dbContext.Set<T>().Add(entity);
  }

  public void Delete<T>(T entity) where T : class
  {
    dbContext.Set<T>().Remove(entity);
  }

  public IEnumerable<T> GetAll<T>() where T : class
  {
    return dbContext.Set<T>();
  }

  public T GetById<T>(IComparable id) where T : class
  {
    return dbContext.Set<T>().Find(id);
  }
```

DAL amb Entity Framework

```
  public bool Exists<T>(IComparable id) where T : class
  {
    return dbContext.Set<T>().Find(id) != null;
  }

  public void Clear<T>() where T : class
  {
    dbContext.Set<T>().RemoveRange(dbContext.Set<T>());
  }

  public void Commit()
  {
    dbContext.SaveChanges();
  }
}
```

Exemple DbContext

```
public class VehicleRentalDbContext : DbContext
{
  public IDbSet<BranchOffice> BranchOffices { get; set; }
  public IDbSet<Reservation> Reservations { get; set; }
  public IDbSet<Category> Categories { get; set; }
  public IDbSet<Person> People { get; set; }
  public IDbSet<Customer> Customers { get; set; }
  public IDbSet<CreditCard> CreditCards { get; set; }

  public VehicleRentalDbContext() : base("Name=VehicleRentalDbConnection")
                                    //connection string name
  {
     /*
     See DbContext.Configuration documentation
     */
     Configuration.LazyLoadingEnabled = true;
     Configuration.ProxyCreationEnabled = true;
}

…
```

Exemple DbContext

```
  protected override void OnModelCreating(DbModelBuilder modelBuilder)
  {
    // Primary keys with non conventional name
    modelBuilder.Entity<Person>().HasKey(p => p.Dni);
    modelBuilder.Entity<Customer>().HasKey(c => c.Dni);
    modelBuilder.Entity<CreditCard>().HasKey(c => c.Digits);

     // Classes with more than one relationship
     modelBuilder.Entity<Reservation>().HasRequired(r => r.PickUpOffice).WithMany(o =>
o.PickUpReservations).WillCascadeOnDelete(false);
     modelBuilder.Entity<Reservation>().HasRequired(r => r.ReturnOffice).WithMany(o =>
o.ReturnReservations).WillCascadeOnDelete(false);
  }

  static VehicleRentalDbContext()
  {
    //Database.SetInitializer<VehicleRentalDbContext>(new CreateDatabaseIfNotExists<VehicleRentalDbContext>());
    Database.SetInitializer<VehicleRentalDbContext>(new
DropCreateDatabaseIfModelChanges<VehicleRentalDbContext>());
    //Database.SetInitializer<VehicleRentalDbContext>(new DropCreateDatabaseAlways<VehicleRentalDbContext>());
    //Database.SetInitializer<VehicleRentalDbContext>(new VehicleRentalDbInitializer());
    //Database.SetInitializer(new NullDatabaseInitializer<VehicleRentalDbContext>());
  }
}
```

Configuració de ORM amb Fluent API

Inicialització de la base de dades

Inicialització de la base de dades

- **CreateDatabaseIfNotExists:** Opció per defecte. Crea la BD si no existeix, segons la configuració. No obstant açò, si canvia el model i s'executa l'aplicació amb aquesta inicialització, llança una excepció.
- **DropCreateDatabaseIfModelChanges:** Si hi ha hagut algun canvi en les classes del model (i per tant canvia l'esquema de BD) esborra la BD existent i crea una nova segons el nou esquema de dades.
- **DropCreateDatabaseAlways:** Esborra la BD existent i crea una nova cada vegada que s'executa l'aplicació. Pot ser útil quan s'està desenvolupant l'aplicació.
- **Custom DB Initializer:** Permet crear un inicialitzador a mesura quan els anteriors no satisfan els requeriments o es desitja executar algun altre procés

Bibliografia

- Martin Fowler (2002). Patterns of Enterprise Application Architecture

- Scott Millet (2015) Patterns, Principles, and Practices of Domain-Driven Design

Material online

- Repository Pattern en MSDN:
- Entity Framework Fluent API - Configuring and Mapping Properties and Types
- Entity Framework Fluent API - Relationships