

DISEÑO DE LA CAPA DE DE PERSISTENCIA

Seminario 5 – Desarrollo de
Software en Visual Studio 2019

Objetivos

- Aprender a diseñar la capa de persistencia aplicando patrones de diseño y principios de separación de capas
- Aplicar los conocimientos de Entity Framework para la implementación de la capa de persistencia

Patrones de diseño para la capa de persistencia (acceso a datos)

- Al diseñar la capa de persistencia estamos interesados en:
 - Abstractar los detalles del mecanismo de implementación a las demás capas
 - Facilitar un hipotético cambio en el mecanismo concreto de persistencia
- En clase de teoría hemos visto dos patrones de diseño apropiados para esos objetivos
 - Patrón DAO
 - Patrón Repository + UoW

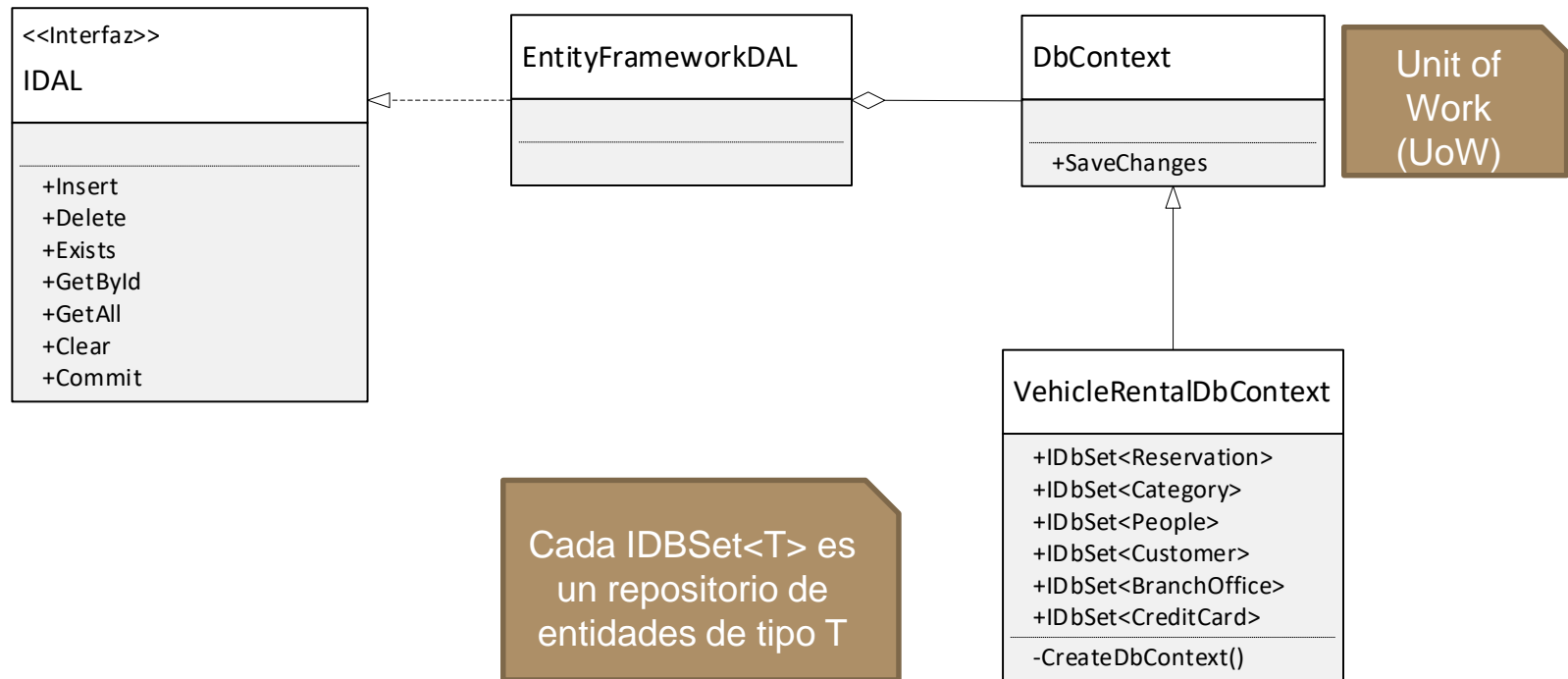
Persistencia y Capa de Acceso a Datos

- Al hablar de **persistencia** nos referimos a toda la infraestructura encargada de persistir y/o recuperar datos en/desde algún mecanismo de almacenamiento, como una base de datos, un archivo o un servicio Web.
- Llamaremos **Capa de Acceso a Datos** (Data Access Layer, abreviado **DAL**) a la parte de la persistencia encargada de dar servicio otras capas (en nuestra arquitectura cerrada de 3 capas dará servicio a la capa de negocio)

Diseño de la capa de persistencia

- En las prácticas de laboratorio vamos a desarrollar una aplicación de escritorio de pequeño tamaño, simplificando la arquitectura aunque usando patrones reconocidos y siguiendo buenas prácticas.
 - Como mecanismo concreto de persistencia usaremos un framework ORM, **Entity Framework**, el cual implementa los patrones **Repository + UoW** para el acceso a datos.
 - Para desacoplar la capa de acceso a datos del mecanismo de implementación usaremos una interfaz pública genérica con todos los métodos necesarios, denominada **IDAL**

Diseño arquitectónico (capa de persistencia)



Interface 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 con 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 con 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();
}
}
```

Ejemplo 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;
    }
    ...
}
```

Ejemplo 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>());
}
}
```

Configuración de
ORM con Fluent API

Inicialización de la
base de datos

Inicialización de la base de datos

- **CreateDatabaseIfNotExists:** Opción por defecto. Crea la BD si no existe, según la configuración. Sin embargo, si cambia el modelo y se ejecuta la aplicación con esta inicialización, lanza una excepción.
- **DropCreateDatabaseIfModelChanges:** Si ha habido algún cambio en las clases del modelo (y por tanto cambia el esquema de BD) borra la BD existente y crea una nueva según el nuevo esquema de datos.
- **DropCreateDatabaseAlways:** Borra la BD existente y crea una nueva cada vez que se ejecuta la aplicación. Puede ser útil cuando se está desarrollando la aplicación.
- **Custom DB Initializer:** Permite crear un inicializador a medida cuando los anteriores no satisfacen los requerimientos o se desea ejecutar algún otro proceso.

Bibliografía

- 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](#)