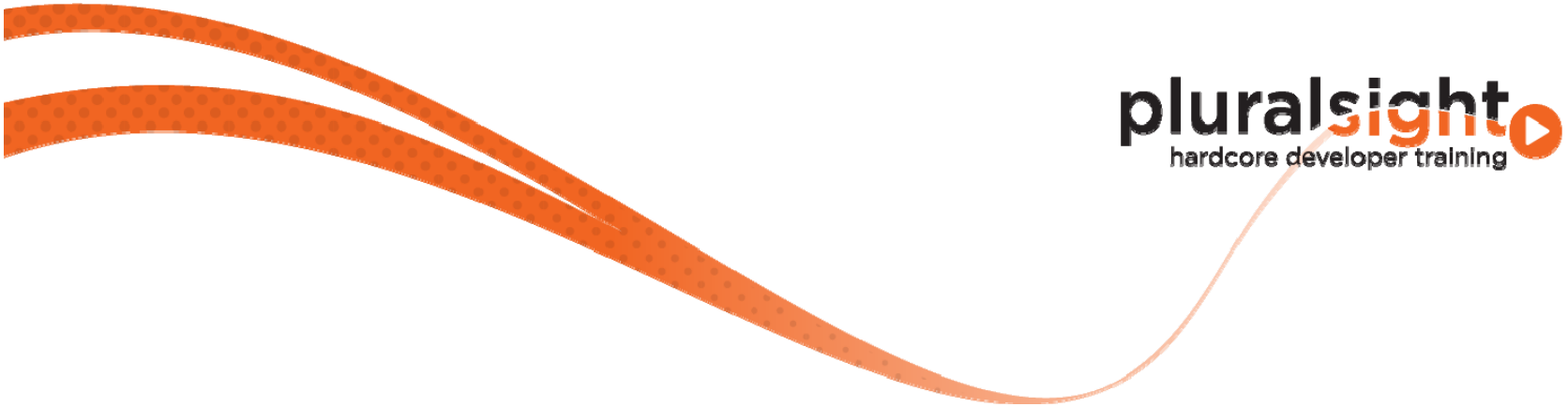
Building End-to-End Multi-Client Service Oriented Applications – *Angular Edition*

Module 05

Data Access

Starting the Business Tier



pluralsight
hardcore developer training

Highlights

- **Entity Framework Code-First**
- **Data Repositories**
  - Base and abstractions
  - Dependency Injection
- **Repository Factory**
  - Abstract factory pattern
- **Usage and unit tests**
- **Additional DTOs (data-transfer-objects)**

Entity Framework Code-First

- **EF Code-First allows manual setup of entities**
  - Exactly what we did
  - Bypasses auto-gen from existing database
    - Can optionally create database elements for you
  - Good for using EF as little or as much as desired
  - You control the code 100% - no excess pollution
- **Business Entities to be used by EF for the Data Access Layer**
- **DB Context class to be setup manually**
  - Defines collections of entities
  - Defines any ORM configuration using EF DSL

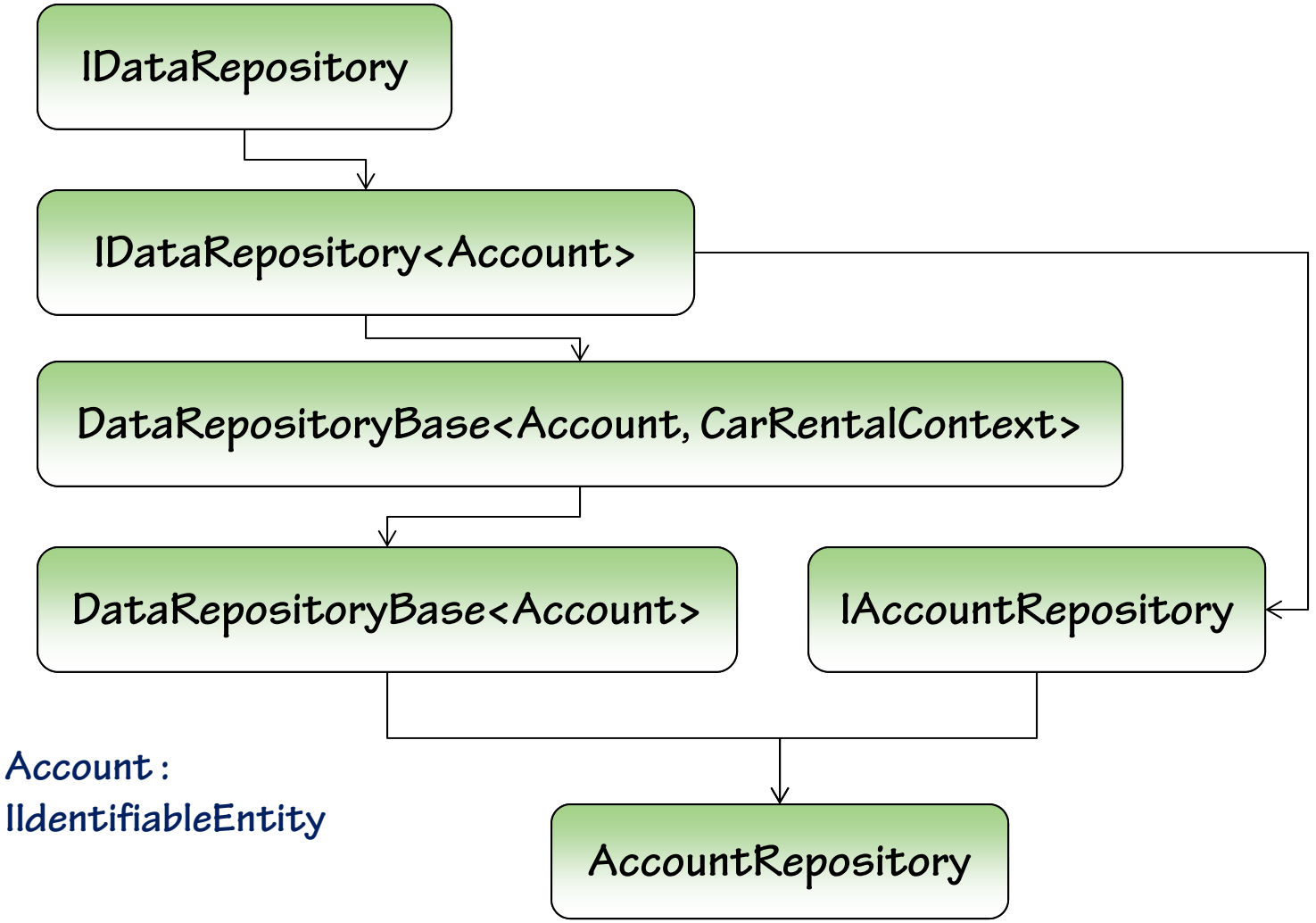
Entity Framework Context Class

```
public class MyContextClass : DbContext
{
   constructor points to connection string

   DbSet<T> properties for tables (T: entity)

   override OnModelCreating method
      uses DbModelBuilder's fluent-interface
      to create ORM
}
```

Data Repositories

- **One data repository per table**
  - Will use Linq-to-Entities in repository methods
- **Need to establish pattern for standard CRUD**
  - Remember **IIdentifiableEntity** ?
  - Base classes/interfaces in core-framework
- **All repository methods s/b interface-defined**
  - Standard CRUD + additional custom methods
- **Data repositories "exported" through MEF**

Data Repository Pattern



Account :
IIdentifiableEntity

Using Data Repositories

- **Data Repositories "exported" through MEF**
- **Client class "imports" data repository interface(s)**
  - Client methods use repository instance
- **Client class provides constructor for testing**
  - Receives instance of data repositories
  - Typically mocked instances from test methods

Data Repository Factory

- **Abstract factory pattern**
- **A repository client class may "import" many repositories**
  - Any given method may need only one or two
  - Excess instantiation during MEF resolve process
- **Factory allows for on-demand repository instantiation**
  - Still uses MEF
- **Client class will only need to import factory interface**
- **Methods will obtain repositories they need from resolved factory**

Using the Data Repository Factory

- **Data Repository factory "exported" through MEF**
- **Client class "imports" data repository factory interface(s)**
  - Client methods obtain repository from factory
  - Client methods use repository instance
- **Client class provides constructor for testing**
  - Receives instance of data repository factory
  - Typically mocked instances from test methods

Additional DTOs

- Used to wrap more than one entity so that one LINQ query can return it.

Summary

- All EF usage encapsulated into DAL
- Everything is interface driven and incorporates DI
- Factory allows for on-demand repository obtainment
- Clients can work only with interfaces, making them testable

- EF code-first Fluent API:
  - http://msdn.microsoft.com/en-us/data/jj591617.aspx

End of module