Dependency Injection

It’s an implementation of the dependency inversion principle.

Require don’t instantiate.

If object A depends on object B then object A should depend on an abstraction of object B and allow object B to be provided to object A rather than object A creating it.

**Why Use It**

Unit Testing

* Mock the data access layer so we don’t actually need a database to test the business layer
* Mock the file system so we don’t actually need to write any files
* Mock an email operation so we don’t actually send any emails.

Modular Design

* We have an object that depends on a third party service (Getting Postal Codes) we later find out we can use a different third party service at a better rate. Since we are not tightly coupled to the implementation we can change the code swap the new code from the old code test it an put it back into production without changing any of the business logic because the business logic only DEPENDS on the ABSTACTION rather than the implementation.

Configurable

* We can create configuration files that drive which abstraction is passed to object A on the fly

Easy upgrade

* We want to stop using LINQ to SQL and use EF we can swap out the implementation without affecting the BL. So if all the BL code base is tightly coupled to the DAL rather than an abstraction then this task could be very difficult

Clean Separation

* Maintains the single responsibility principle

--This insures that any given implementation is only responsible to the code base that uses it

Code Re-use

* Any implementation that satisfies the abstraction can be RE-USED by any application that needs to use that interface

Data Access Implementations

Data Access Interfaces

Database

Web App