Intro to Dependency Injection & Inversion of Control

What I intend to do…

- Present Dependency Injection and Inversion of Control in an understandable fashion
- Present each topic at a detailed but comprehendible level
- Give you the resources used in this talk so you can reference them in the future.

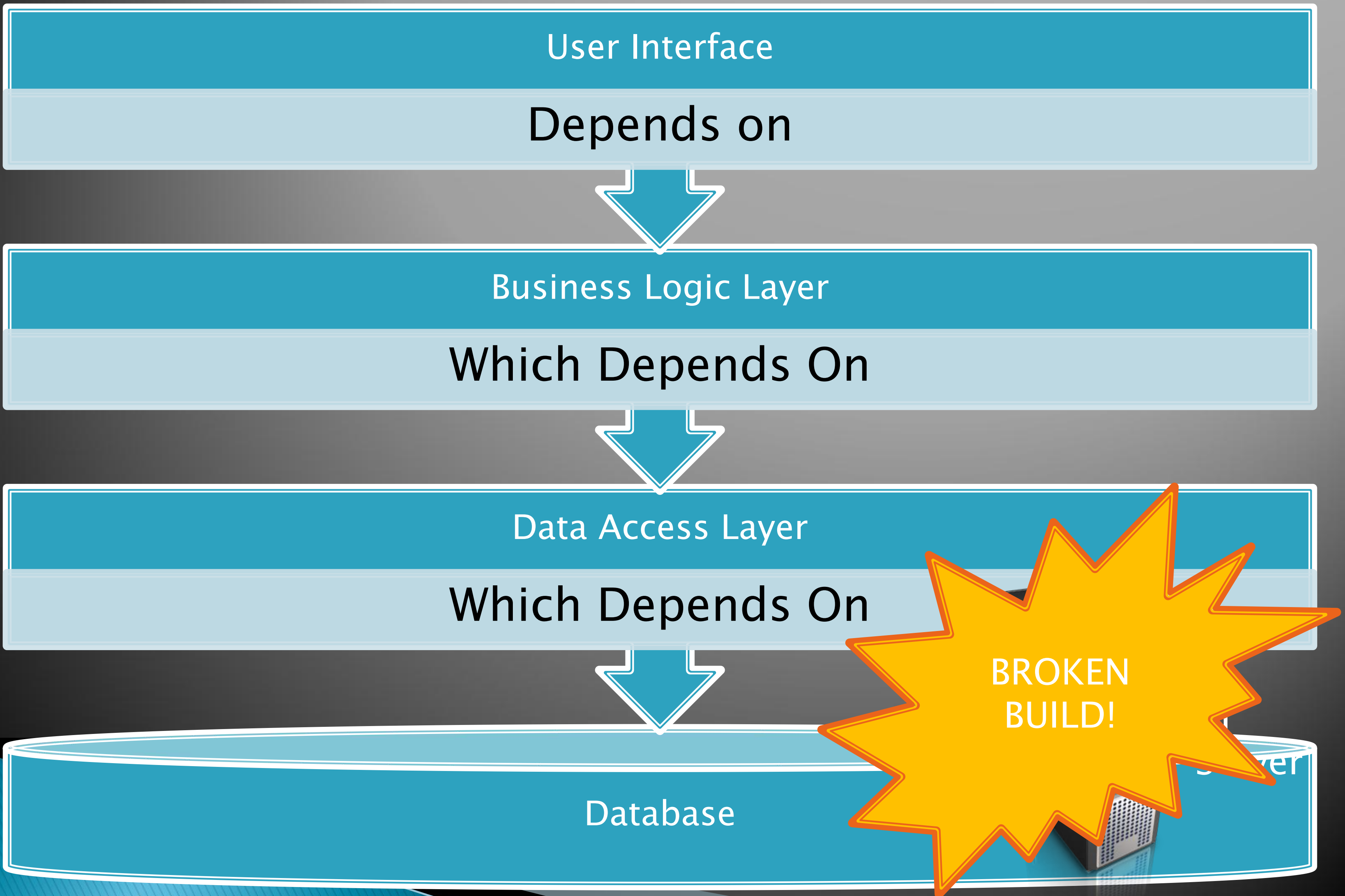
Agenda

- What is a Dependency?
- Dependency Injection Pros/Cons
- Simple Application Architecture
- Example Application High Level Architecture
- Demonstration 1
  - Identifying and Breaking dependencies
- What is Inversion of Control
- Demonstration 2
  - Custom Dependency Container
  - Introducing Microsoft Unity Container
- Questions

What is a “Dependency”?

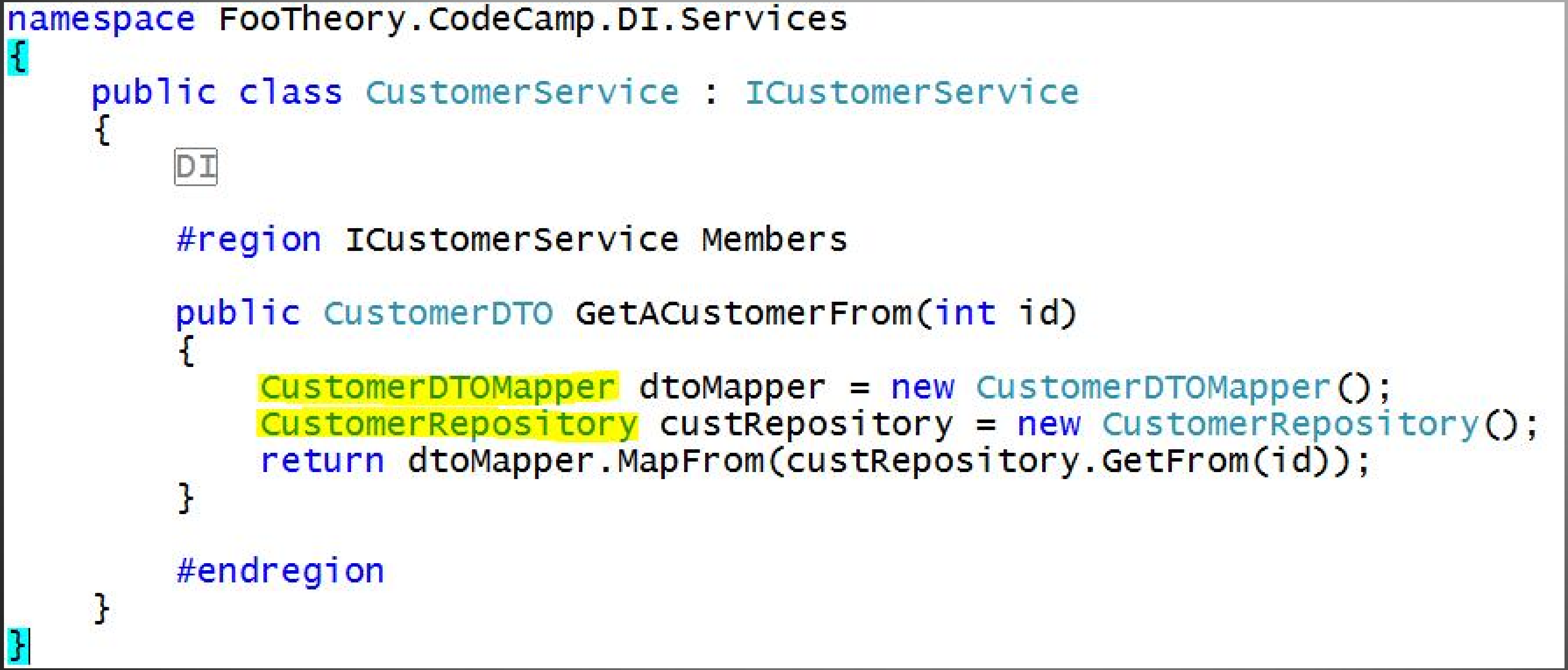
- Some common dependencies include:
  - Application Layers
    - Data Access Layer & Databases
    - Business Layer
  - External services & Components
    - Web Services
    - Third Party Components
  - .NET Framework Components
    - File Objects (File.Delete(…), Directory.Exists(…))
    - Web Objects (HttpContext, Session, Request, etc)

Dependencies at a Very High Level



Get a Resource Involved…

Example of a Dependency



```
namespace FooTheory.CodeCamp.DI.Services
{
    public class CustomerService : ICustomerService
    {
        DI

        #region ICustomerService Members

        public CustomerDTO GetACustomerFrom(int id)
        {
            CustomerDTOMapper dtoMapper = new CustomerDTOMapper();
            CustomerRepository custRepository = new CustomerRepository();
            return dtoMapper.MapFrom(custRepository.GetFrom(id));
        }

        #endregion
    }
}
```

What problems do dependencies create?

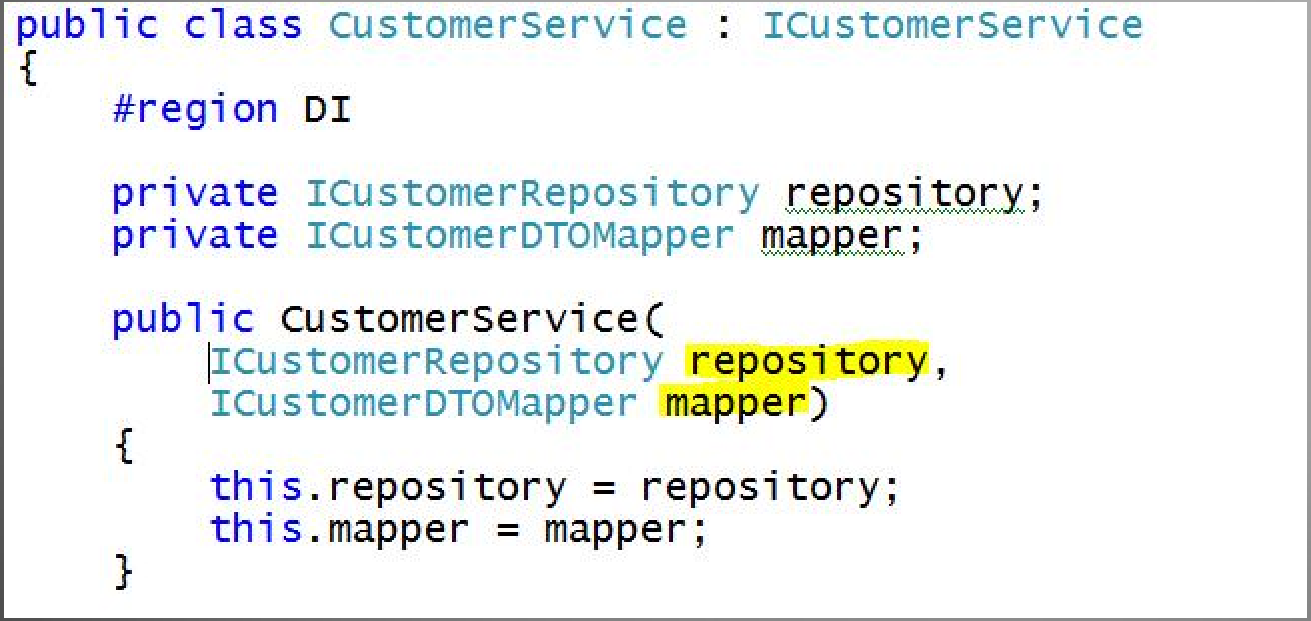
- Code is tightly coupled
- Difficult to isolate when testing
- Difficult to maintain
  - If I change ComponentX how do I know what else it will affect? Did I break anything?
    - If tests are in place they can be your safety net

What is Dependency Injection?

- The ability to supply (inject) an external dependency into a software component.

- Types of Dependency Injection:
  - Constructor (Most popular)
  - Setter
  - Method

Constructor Injection

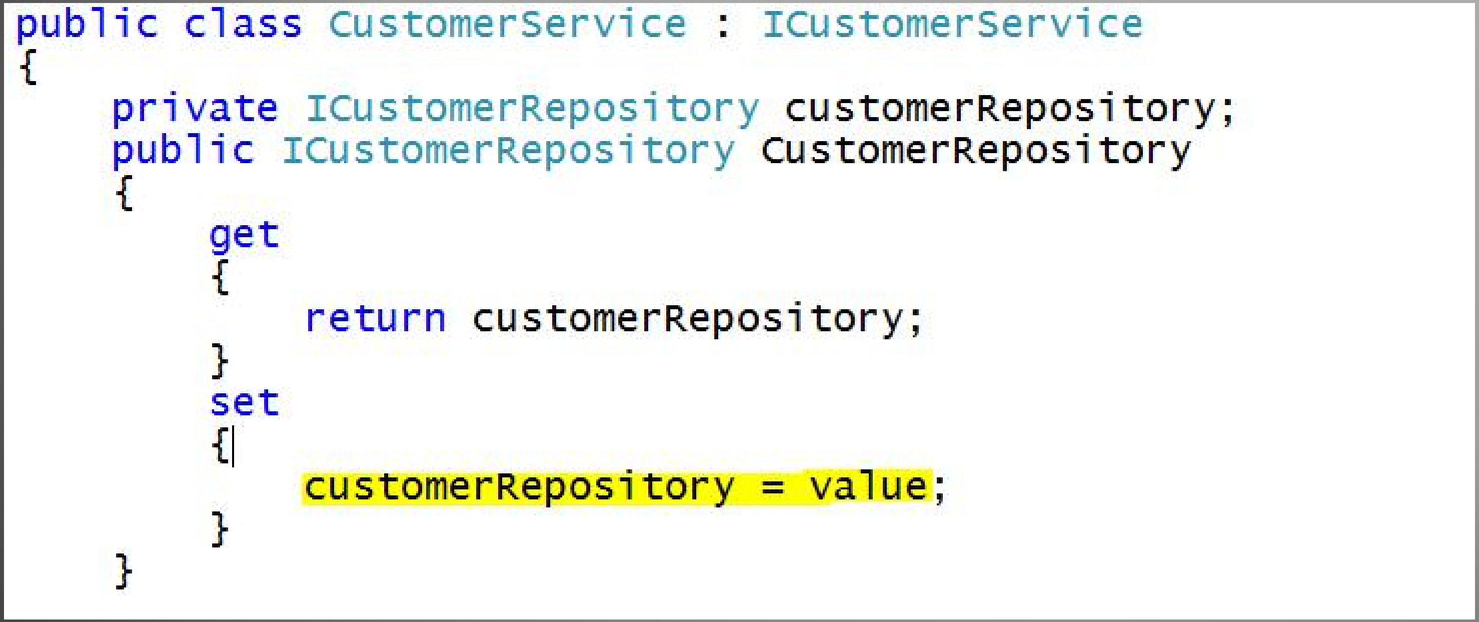


```
public class CustomerService : ICustomerService
{
    #region DI

    private ICustomerRepository repository;
    private ICustomerDTOMapper mapper;

    public CustomerService(
        ICustomerRepository repository,
        ICustomerDTOMapper mapper)
    {
        this.repository = repository;
        this.mapper = mapper;
    }
```

Injecting a ICustomerRepository and a ICustomerDTOMapper through the constructor.

Note: This is the most popular type of injection.

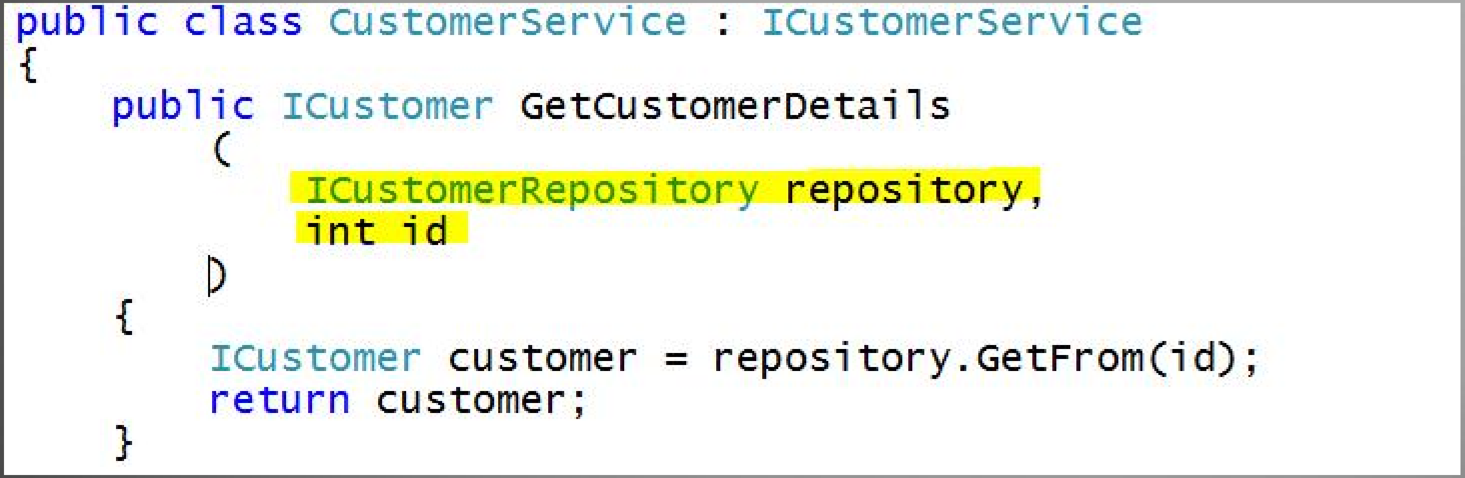
Setter Injection



```
public class CustomerService : ICustomerService
{
    private ICustomerRepository customerRepository;
    public ICustomerRepository CustomerRepository
    {
        get
        {
            return customerRepository;
        }
        set
        {
            customerRepository = value;
        }
    }
```

Injecting a ICustomerRepository through the setter.

Method Injection



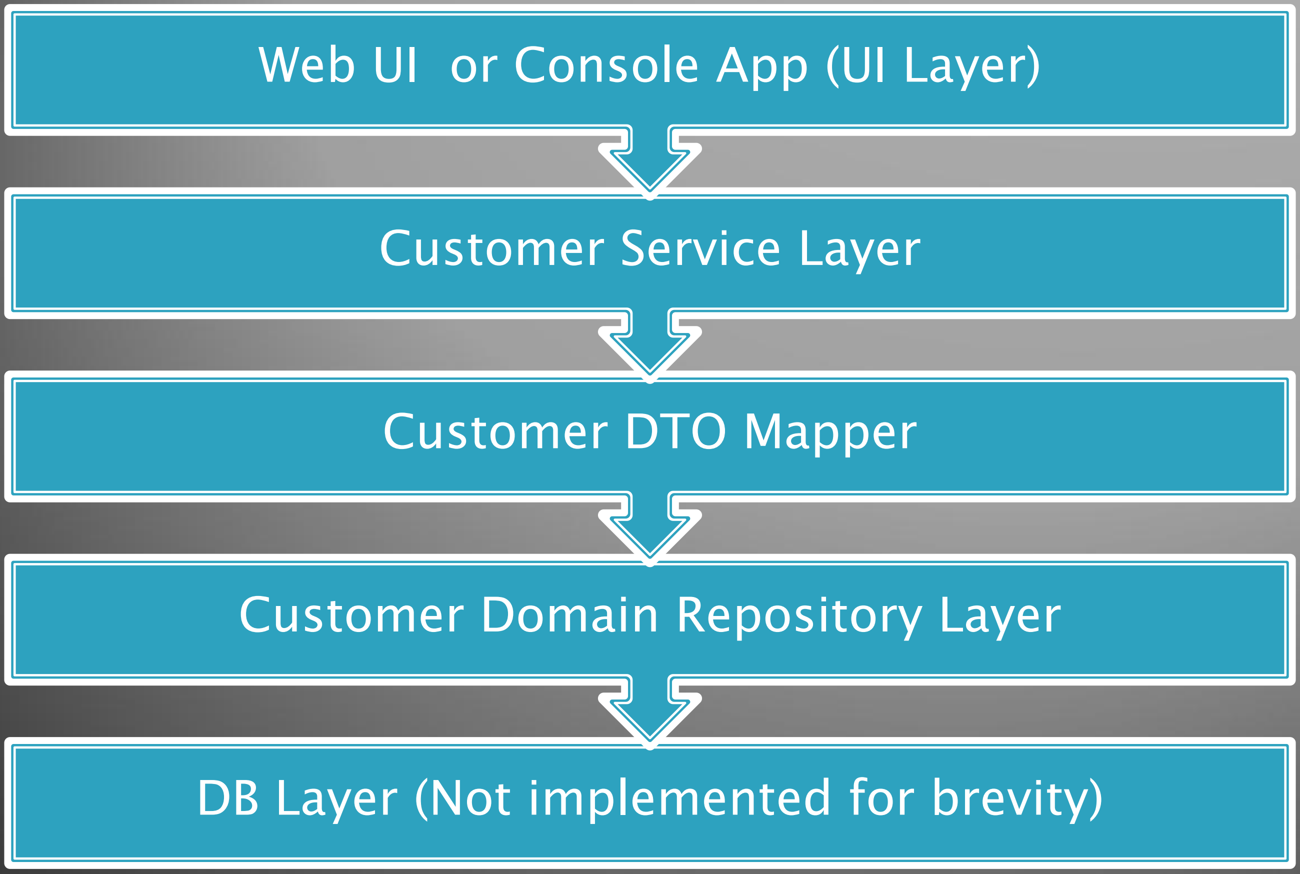
```
public class CustomerService : ICustomerService
{
    public ICustomer GetCustomerDetails
        (
            ICustomerRepository repository,
            int id
        )
    {
        ICustomer customer = repository.GetFrom(id);
        return customer;
    }
```

Injecting a ICustomerRepository as well as an integer dependency.

Dependency Injection Pros & Cons

- Pros
  - Loosely Coupled
  - Increases Testability (A LOT!)
  - Separates components cleanly
  - Allows for use of Inversion of Control Container
- Cons
  - Increases code complexity
  - Some Jr. Developers find it difficult to understand at First
  - Can Complicate Debugging at First
  - Complicates following Code Flow

Overview of Example Application


Web UI or Console App (UI Layer)
Customer Service Layer
Customer DTO Mapper
Customer Domain Repository Layer
DB Layer (Not implemented for brevity)

Demonstration

- Lets See Some Code…

What is Inversion of Control

- Sometimes referred to as Dependency Inversion Principle (DIP)
  - The principle states that high level or low level modules should not depend upon each other, instead they should depend upon abstractions.
- Specific implementations (object instances) are deferred to a higher level of abstraction of control.
  - Examples:
    - Parent class(es)
    - A Container
- Referred to as the “Hollywood Principle”
  - “Don’t call us, we will call you.”

IoC Demonstration

- The best example is to see it in code.