



An Introduction to Dependency Injection and IoC Containers

WHY YOU SHOULD CARE ABOUT DEPENDENCY INJECTION

About Me



- Jeff Barnes
- .NET Software Craftsman @ DAXKO
- Microsoft MVP in Connected Systems
- ALT.NET Supporter

- jeff@jeffbarnes.net
- <http://jeffbarnes.net/blog>
- http://twitter.com/jeff_barnes






Disclaimer

I do not claim to be an expert in anything!

I'm here to simply share things that I have learned about topics on which I am quite passionate.



Question everything I say and form your own opinions based on the information!



Agenda

- What is dependency injection?
- How does it work?
- Why would you want to use it?
- What is an IoC container?
- What does it buy you?
- How should you use it?

DI vs IoC – What's the Diff?

- Dependency Injection != Inversion of Control
 - IoC is an abstract concept about inverting the flow of control within an application/system
 - The Hollywood Principle
 - Don't call us, we will call you.
- DI is a specific form of IoC
 - Creation of dependencies is inverted
- The terms are often used interchangeably

So, what is DI?

- Rather than a class directly instantiating its own dependencies, it is someone else's responsibility to provide the dependencies to the given class.
- In a nutshell:
 - It is all about externalizing dependencies and coordinating how they are provided to a class

Why should you care?


- Reduces coupling
- Improves flexibility
- Improves testability and makes it easier to use mocks for isolating the logic under test





How Does It Work?

- Open Closed Principle
 - Dependency Inversion Principle

 - Dependencies are provided to the given class
 - Constructor Injection
 - Field Injection
 - Property Injection
 - Method Injection
- 



OCP Refresher

- According to Uncle Bob:

Classes should be open for extension, but closed for modification.





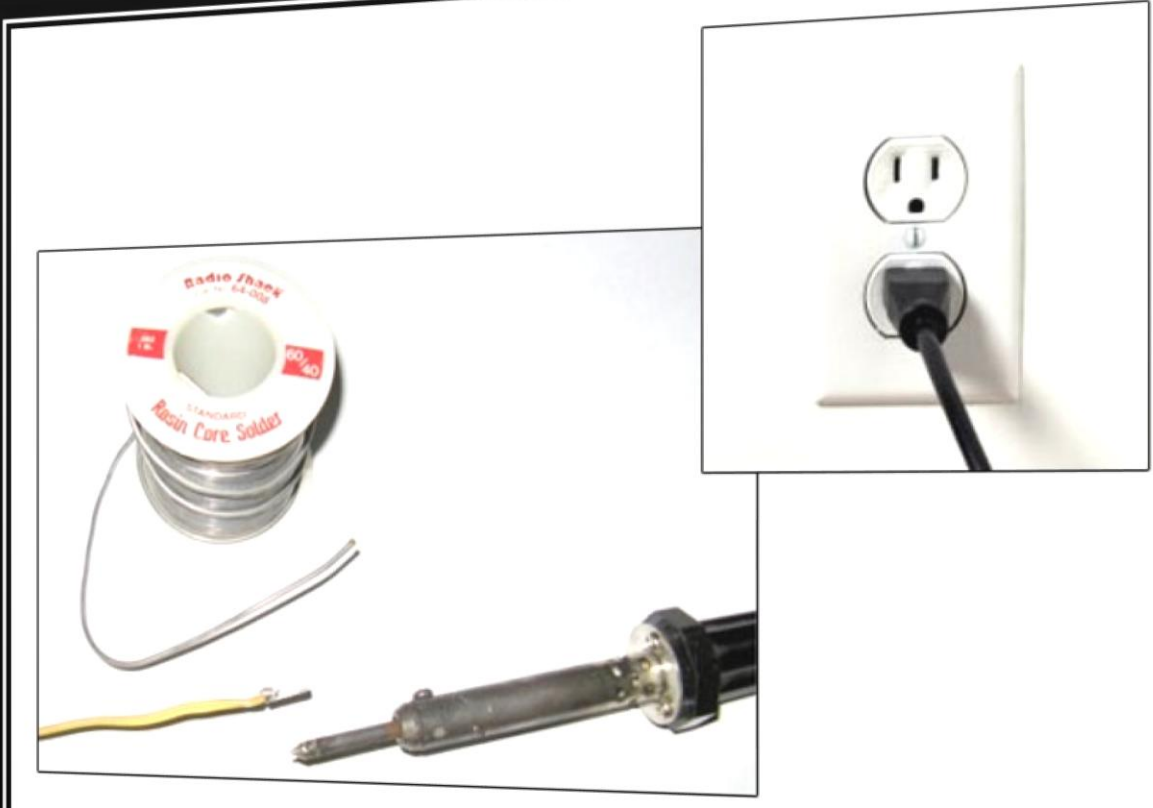
OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat

DIP Refresher

- According to Uncle Bob:
High level modules should not depend upon low level modules. Both should depend upon abstractions.

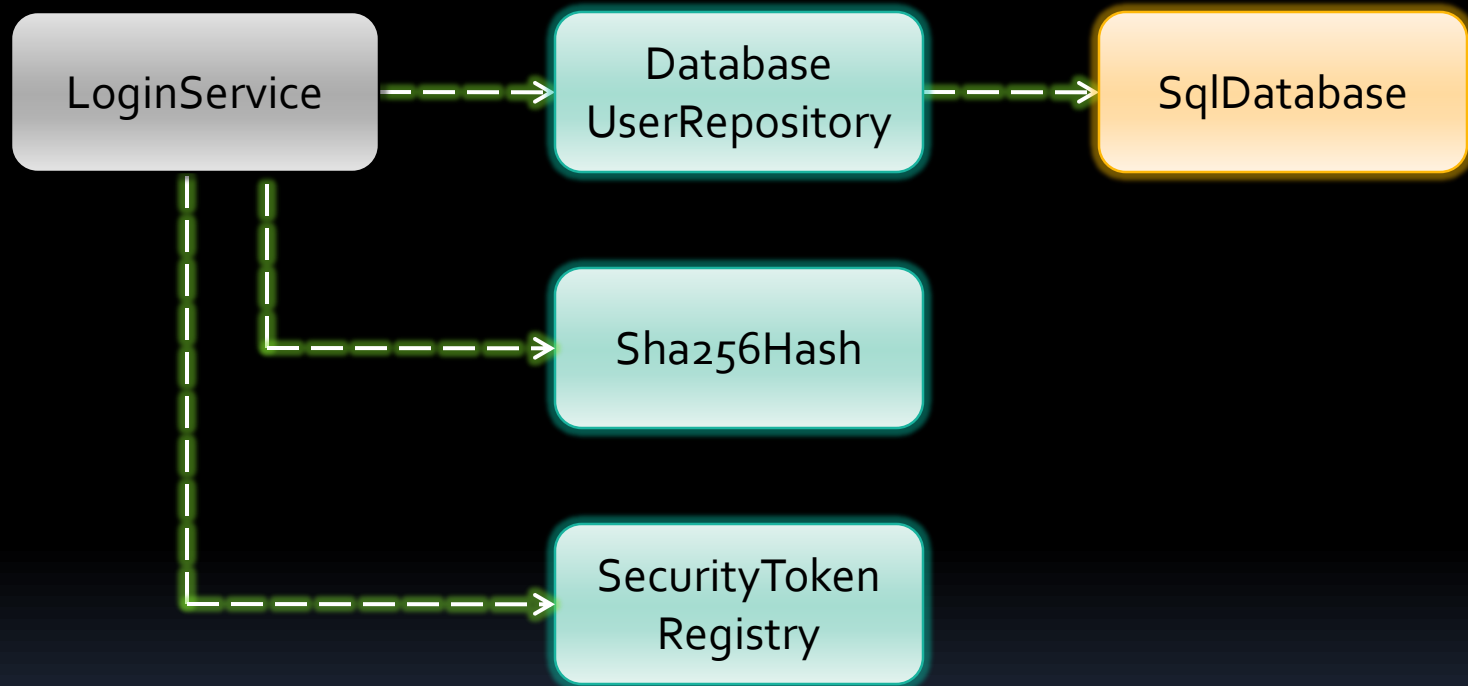
Abstractions should not depend upon details.
Details should depend upon abstractions.



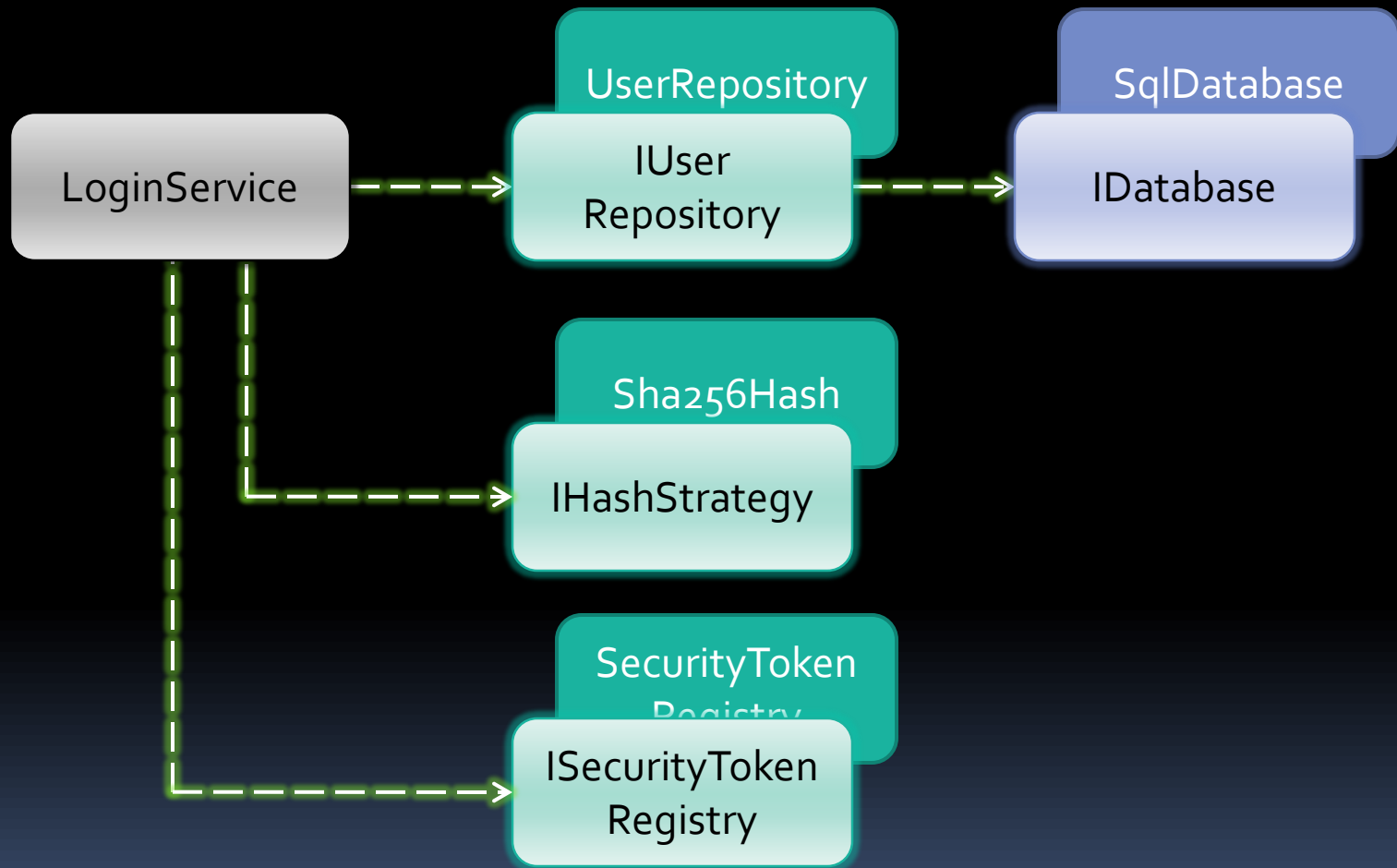
DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

Without Dependency Injection



With Dependency Injection



Deep Dependencies Suck...

- When handled manually...
- Dependencies with deep hierarchies can be painful for the caller to setup and supply to the given class.
- How can we make this better:
 - Poor Man's Dependency Injection
 - Service Locator
 - IoC Container

Poor Man's DI

- Relies on default (empty) constructors that chain to constructor overloads which accept the required dependencies.
- Provides default dependencies with option to override them by the caller.
- A step in the right direction, but still couples the class to the dependencies



Service Locator

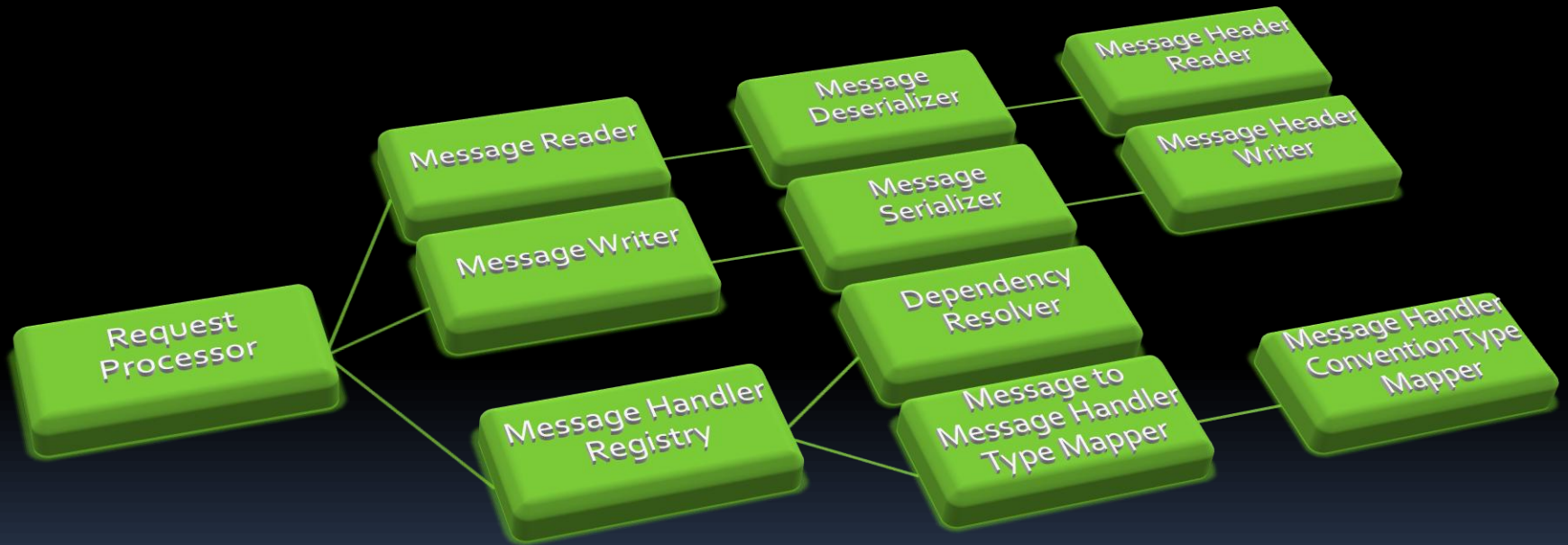
- Relies on a globally accessible class that can be used to lookup the instance that should be used for the given type.
- Gets the job done, but still not ideal.
 - Necessitates some coupling
 - Necessitates a dependency on the service locator
 - Requires mocking the service locator in tests



IoC Container


- Separate component that is responsible for maintaining all of the dependencies and their corresponding lifecycles.
- The sweet spot, but comes with complexity:
 - Requires bootstrapping/configuration
 - Can be more difficult to see the big picture
 - Makes dependency injection much easier (when used correctly)

The Beauty of IoC Containers





Popular .NET IoC Containers

- StructureMap
 - Castle Windsor / Microkernel
 - Ninject
 - Unity
 - Spring.NET
 - Autofac
 - PicoContainer
- 

IoC Container Dos and Donts

- Do
 - Leverage your container near the top level
 - Favor programmatic registration
 - Put it to work **for** you
- Don't
 - Excessively use the container as a service locator
 - Go crazy with XML configurations
 - Put it to work **against** you

