.NET Developer Toolkit

DEPENDENCY INJECTION IN .NET

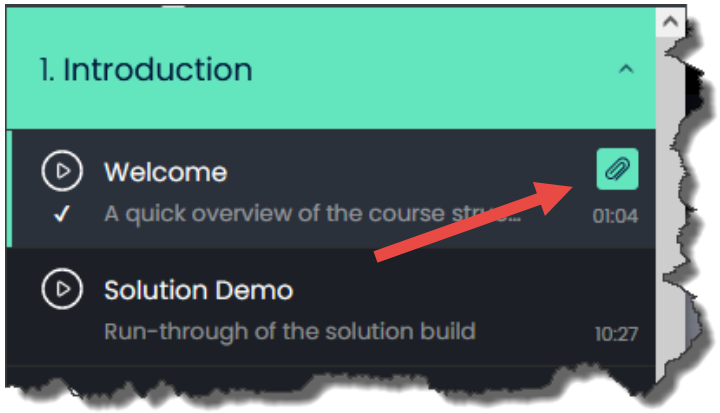
What we'll cover

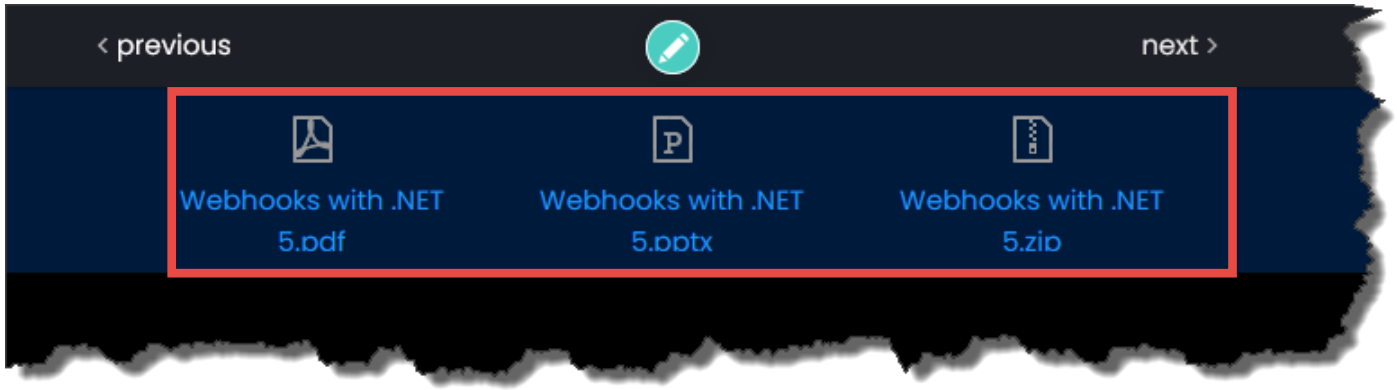
- Overview of Dependency Injection (theory)
- Code Example 1 – the basics
- Dependency Injection in .NET (theory)
- Code Example 2 – a familiar pattern
- Service Lifetimes (theory)
- Code Example 3 – out of scope!
- Extra Credit Code Example 4 – Service Lifetimes deep-dive

Ingredients

- .NET 6 SDK (free)
- VSCode (free)
- Web Browser or API Client (Postman or Insomnia)

Course Downloads

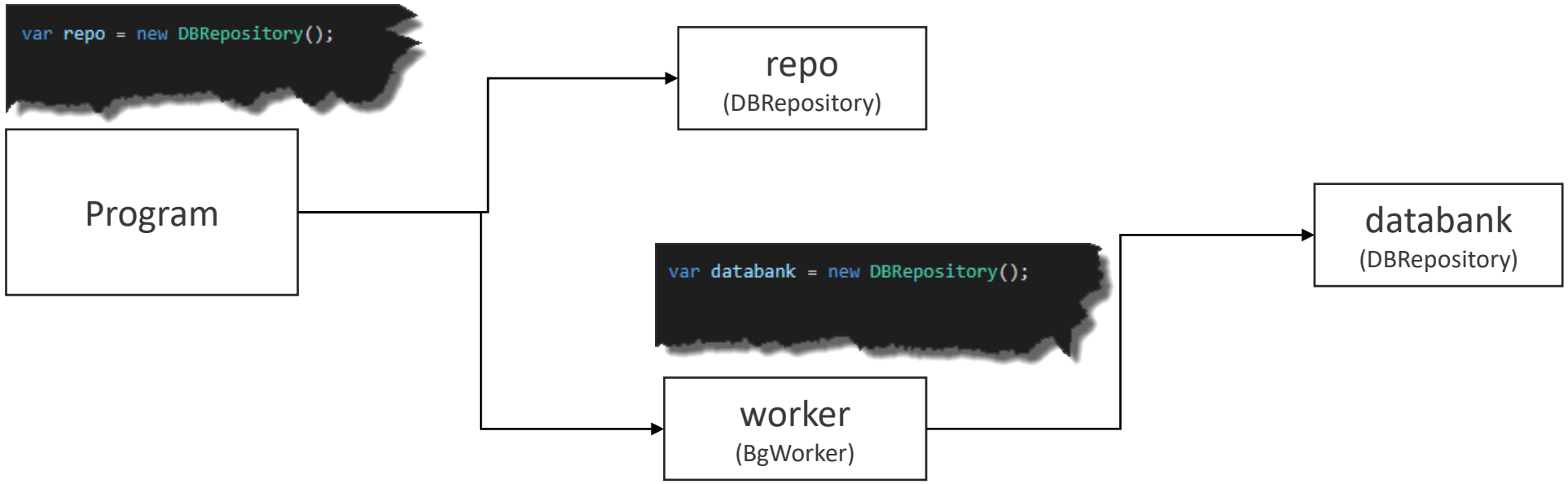




The Basics

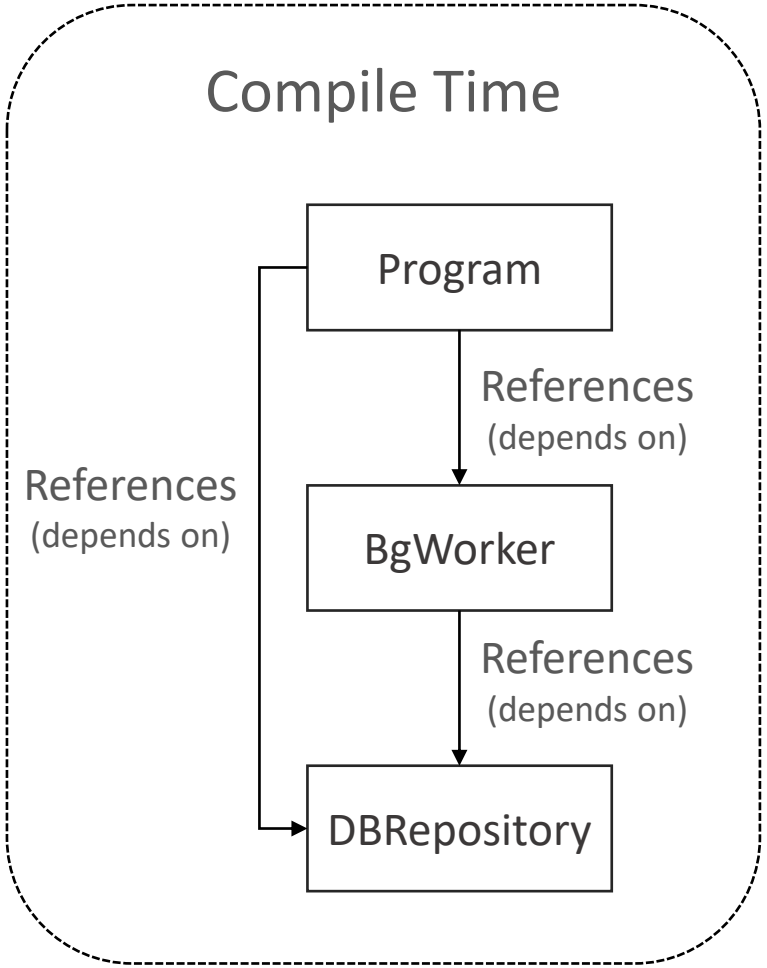
What is Dependency Injection?

Life Before Dependency Injection



- Program *depends* on DBRepository
- Program *depends* on BackgroundWorker
- BackgroundWorker *depends* on DBRepository

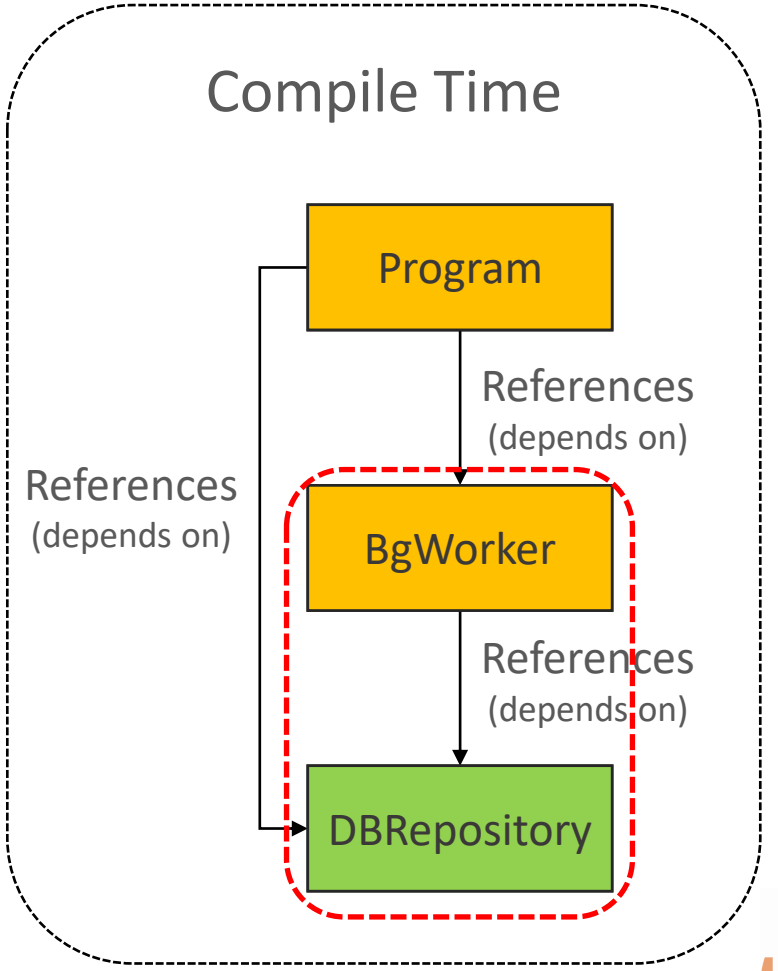
Direct Dependency Graph*



* Referenced from Microsoft “Architectural Principals”

What’s wrong with that?

- We can have long dependency chains
- If we want to change the implementation we must do it in many places
- Difficult to unit test
  - We just want to test the “unit”



Dependency Injection (DI)

- Dependency Injection is a Design Pattern
- Used predominantly in Object Oriented Programming
- Aims to allow us to develop “loosely-coupled” code
- With primary aim of making our code more maintainable

What does this solve?

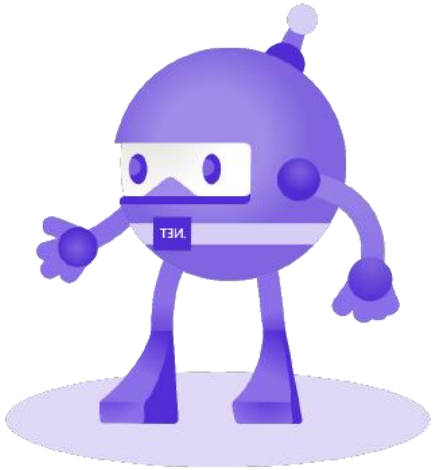
- We can swap implementations with ease
- We break the long dependency chains
- Unit testing becomes easier by removing direct dependencies and using an abstraction

It avoids “tight-coupling”

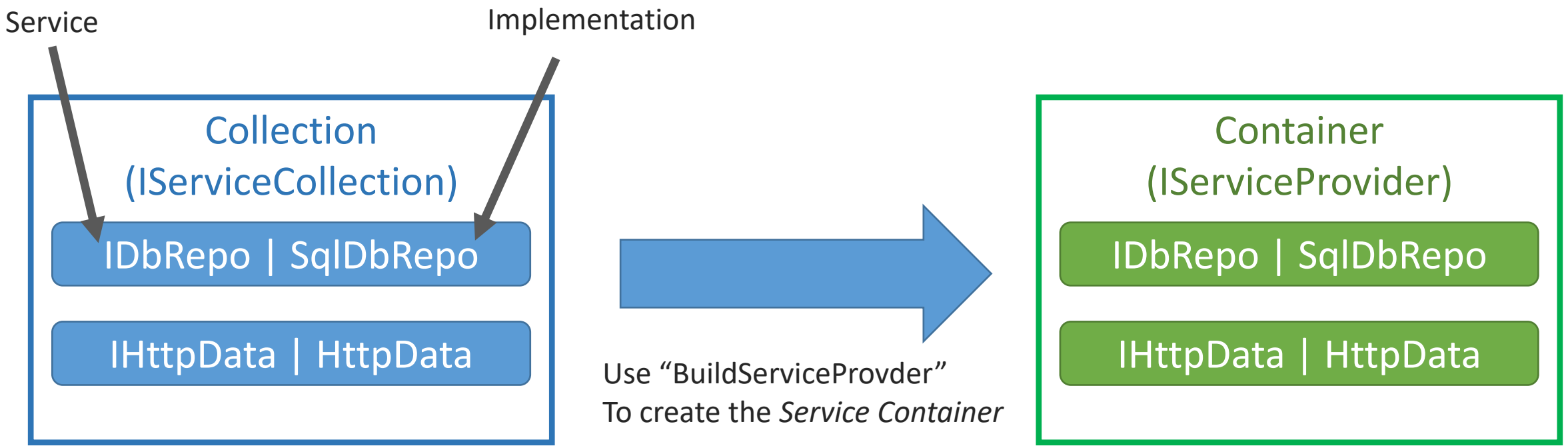
Let's Code!

Dependency Injection in .NET

Terms & Concepts in .NET



Key Terms an Concepts



- This set up is done at application start up
- The DI framework (*container*) provides an instance of the dependency
- The service injected via the constructor of the class where it is used

Service Registration Methods

Method	Auto Object Disposal	Multiple Implementations	Pass Args
`services.AddSingleton<IAppHost, AppHost>();`	Yes	Yes	No
`services.AddSingleton<AppHost>();`	Yes	No	No
`services.AddSingleton<IAppHost>(sp => {new AppHost(99)})`	Yes	Yes	Yes
`Services.AddSingleton<IAppHost>(new AppHost(99));`	No	Yes	Yes
`Services.AddSingleton(new AppHost(99));`	No	No	Yes

https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection

Let's Code!

Service Lifetimes

Service Lifetimes

Transient

- Created each time they are requested from the Service Container
- Best for lightweight stateless services
- Register as transient where possible
- Multi-threading, & memory leaks not a large a consideration

Scoped

- Services are created per “Scope”
- For web apps created for each *request*
- Best for services that need to persist for a request

Singleton

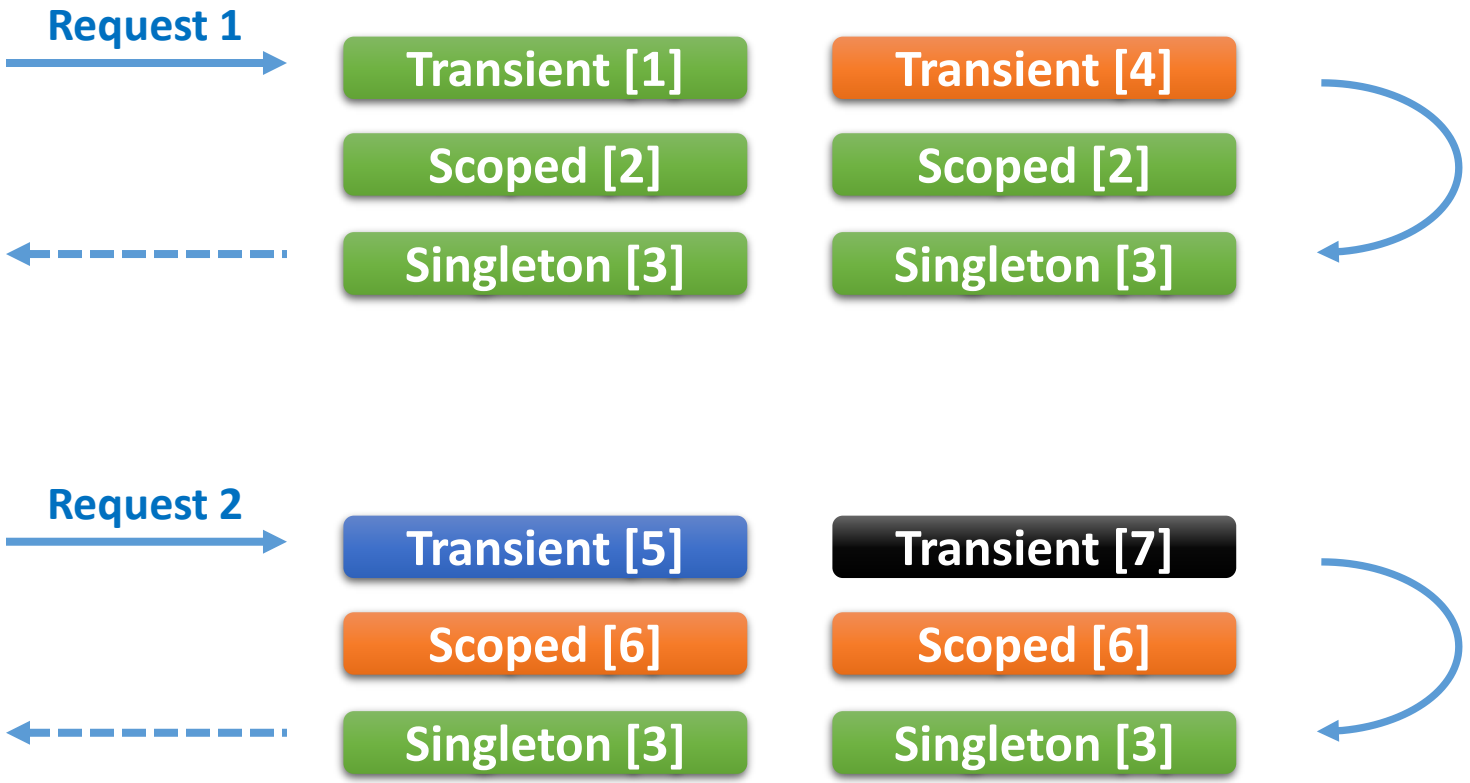
- Created per Container (the first time they are requested)
- Subsequent requests to the Service Container will serve up the same instance
- Used to maintain state
- Exist for the application lifetime
- Need to consider multi-threading considerations

In Pictures

ITransient | Transient

IScoped | Scoped

ISingleton | Singleton

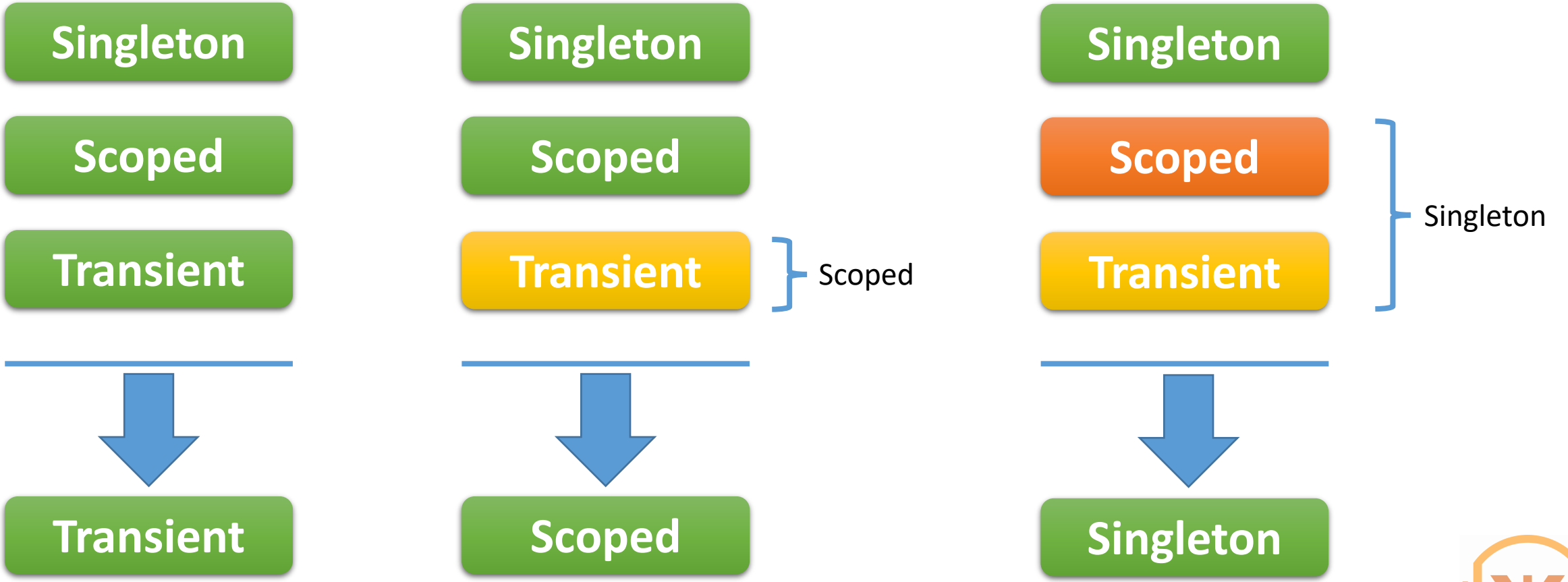


Let's Code!

General Rule

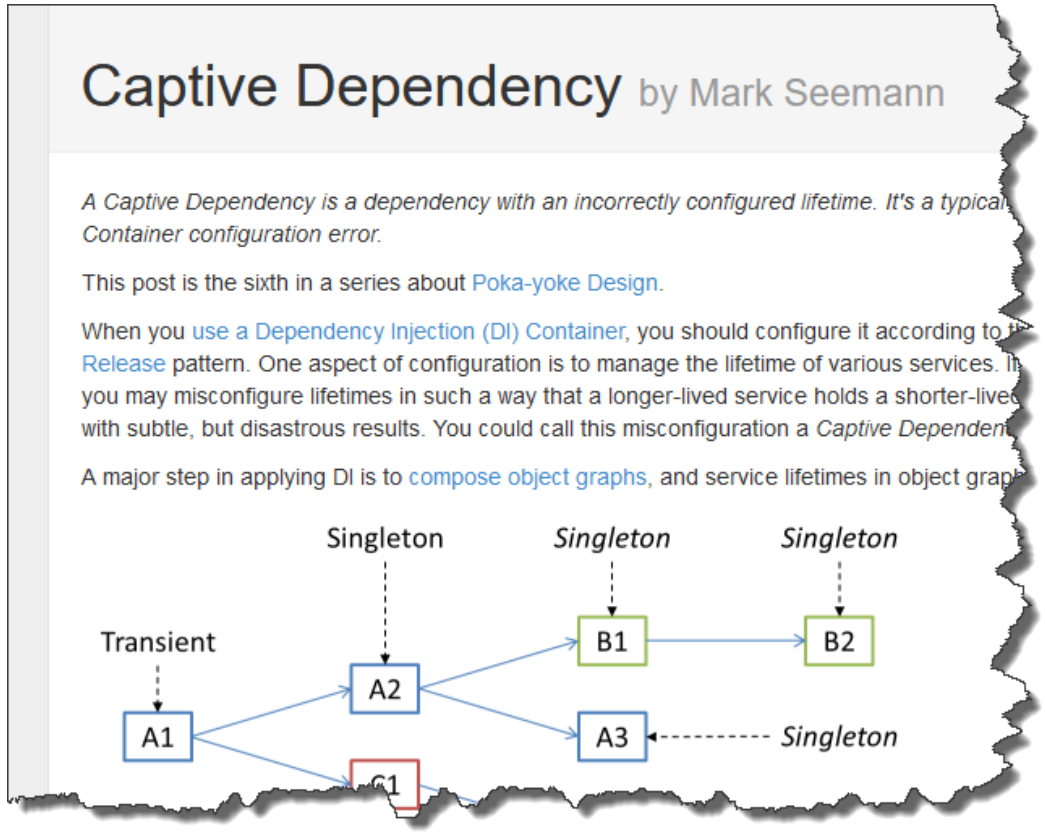
Do not inject services with a shorter lifetime into services with a longer lifetime..

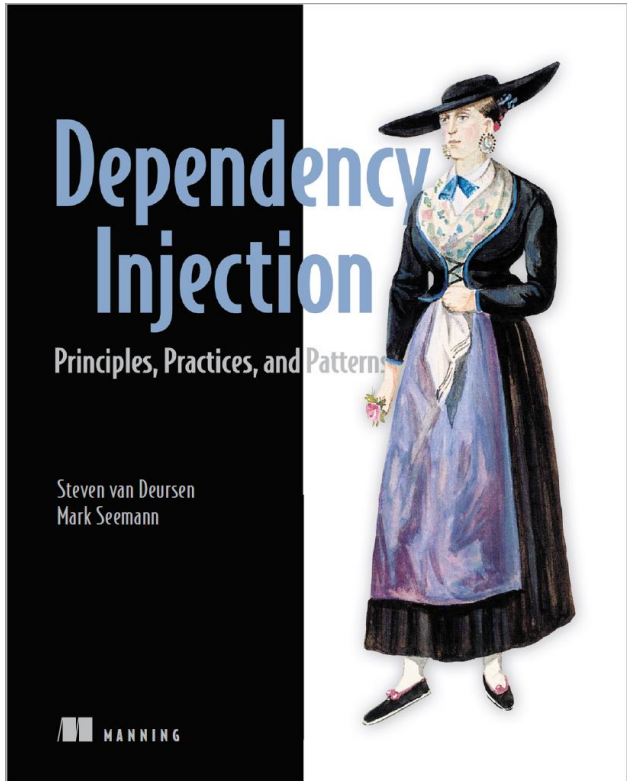
Service Consumption



Captive Dependencies

https://blog.ploeh.dk/2014/06/02/captive-dependency/





Let's Code!

(again)


CODE
SMELL!

Bonus Code

Lifetime Example