Malte Clasen

Dependency Injection Grundlagen

Anwendung: Rezept-DB

Streuselkuchen

Foto von Malte

Kategorien

- Kuchen und Torten
- Kleingebäck

Zutaten

Teig

400	g	Mehl
75	g	Margarine
75	g	Zucker
150	ml	Soja-Reis-Drink
1	Pk	Hefe, trocken
1	Pk	Vanillezucker
		Salz

Streusel

300	g	Mehl
175	g	Margarine
300	g	Zucker
		Zimt

Zubereitung

Teig

Mehl in eine Schüssel geben und eine Kuhle hineindrücken. Zucker, Hefe und *lauwarmen* Soja-Reis-Drink hineingeben. Mit einem Küchentuch zudecken und 15 Minuten lang gehen lassen. Margarine zerlassen und ebenfalls in die Kuhle geben. Von innenheraus mit dem restlichen Mehl verkneten. Nochmals 15 Minuten lang zugedeckt gehen lassen. Auf einem mit Backpapier ausgelegten Blech ausrollen und wieder 15 Minuten lang gehen lassen.

Streusel

Mehl, Zucker und Zimt vermischen, Margarine zerlassen und alles verkneten. Streusel auf dem ausgerollten, gegangenen Teig **gleichmäßig** verteilen und ca. 20 Minuten bei 180°C Umluft backen.

Repository Interface

```
public interface IRecipeRepository
{
    Recipe Get(Guid id);
    void Put(Recipe recipe);
    long Count { get; }
}
```

Test: Throws on invalid Get

```
[TestMethod]
public void ThrowsOnInvalidGet()
{
    var repository = new RecipeRepositoryUsingNew();

    repository.Invoking(r => r.Get(Guid.NewGuid()))
        .ShouldThrow<ArgumentException>();
}
```

Test: Logs invalid Get

```
[TestMethod]
public void LogsInvalidGet()
{
    var repository = new RecipeRepositoryUsingNew();

    repository.Invoking(r=>r.Get(Guid.NewGuid()))
        .ShouldThrow<ArgumentException>();

    repository.Log.Get().Should().HaveCount(1);
}
```

Logging

```
public Recipe Get(Guid id)
{
    Recipe result;
    if (!_recipes.TryGetValue(id, out result))
    {
        Log.Error(Clock.Now, "id not found");
        throw new ArgumentException("id not found");
    }
    return result;
}
```

new Log, new Clock

```
public RecipeRepositoryUsingNew()
{
    Clock = new SystemClock();
    Log = new Log();
}
```

Test: Logs time on invalid Get

```
[TestMethod]
public void LogsTimeOnInvalidGet()
{
    var repository = new RecipeRepositoryUsingNew();

    repository.Invoking(r => r.Get(Guid.NewGuid()))
        .ShouldThrow<ArgumentException>();

    var entry = repository.Log.Get().First();
    Math.Abs((entry.Time - DateTime.Now)
        .TotalSeconds.Should().BeLessThan(1);
}
```

Clock-Interface

```
public interface IClock
{
    DateTime Now { get; }
}
```

System Clock

```
public class SystemClock : IClock
{
    public DateTime Now
    {
      get { return DateTime.Now; }
    }
}
```

Static Clock

```
public class StaticClock : IClock
{
    public DateTime Now { get; set; }

    public StaticClock()
    {
      Now = new DateTime(2013, 1, 1);
    }
}
```

Clock via Factory

```
public RecipeRepositoryUsingFactory()
{
    var logFactory = new LogFactory();
    Log = logFactory.Get();

    var clockFactory = new ClockFactory();
    Clock = clockFactory.Get();
}
```

Clock Factory

```
public IClock Get()
{
    //todo: Context detection Voodoo to select clock type
    return new SystemClock();
}
```

Clock via Abstract Factory

```
public RecipeRepositoryUsingAbstractFactory
    (ILogFactory logFactory, IClockFactory clockFactory)
{
    Log = logFactory.Get();
    Clock = clockFactory.Get();
}
```

Clock Factory Interface

```
public interface IClockFactory
{
    IClock Get();
}
```

Test: Abstract Factory

```
public void LogsTimeOnInvalidGet()
{
    ILogFactory logFactory = new LogFactory();
    IClockFactory clockFactory =
      new StaticClockFactory();
    var repository =
      new RecipeRepositoryUsingAbstractFactory
      (logFactory, clockFactory);
    var referenceTime = repository.Clock.Now;
...
    repository.Log.Get().First().Time
      .Should().Be(referenceTime);
}
```

Clock via Service Locator

```
public RecipeRepositoryUsingServiceLocator()
{
    _log = ServiceLocator.Log;
    _clock = ServiceLocator.Clock;
}
```

Service Locator Interface

```
public static class ServiceLocator
{
    public static ILog Log { get; set; }
    public static IClock Clock { get; set; }
}
```

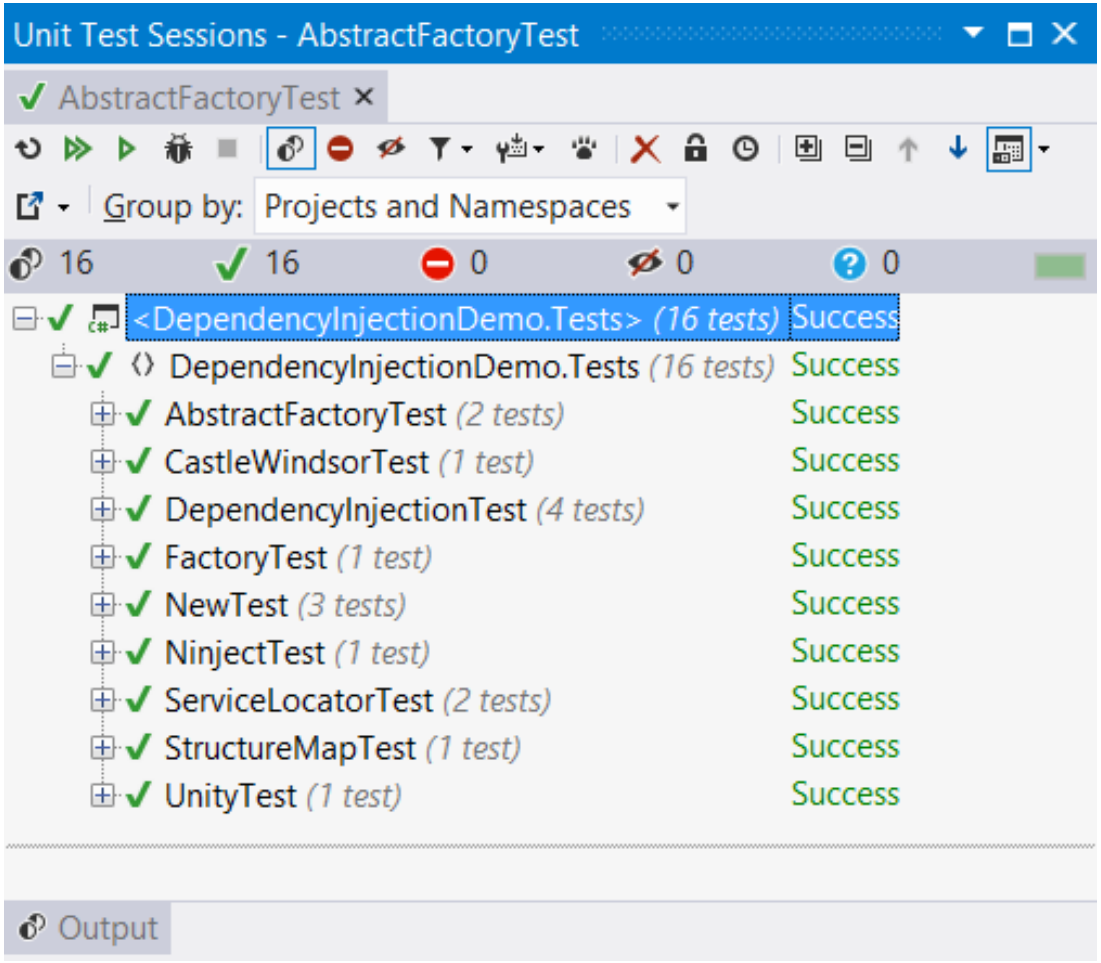
Test: Initialisierung

```
[TestMethod]
public void LogsInvalidGet()
{
    ServiceLocator.Log = new Log();
    ServiceLocator.Clock = new StaticClock();
    var repository =
        new RecipeRepositoryUsingServiceLocator();

    repository.Invoking(r => r.Get(Guid.NewGuid()))
        .ShouldThrow<ArgumentException>();

    ServiceLocator.Log.Get().Should().HaveCount(1);
}
```

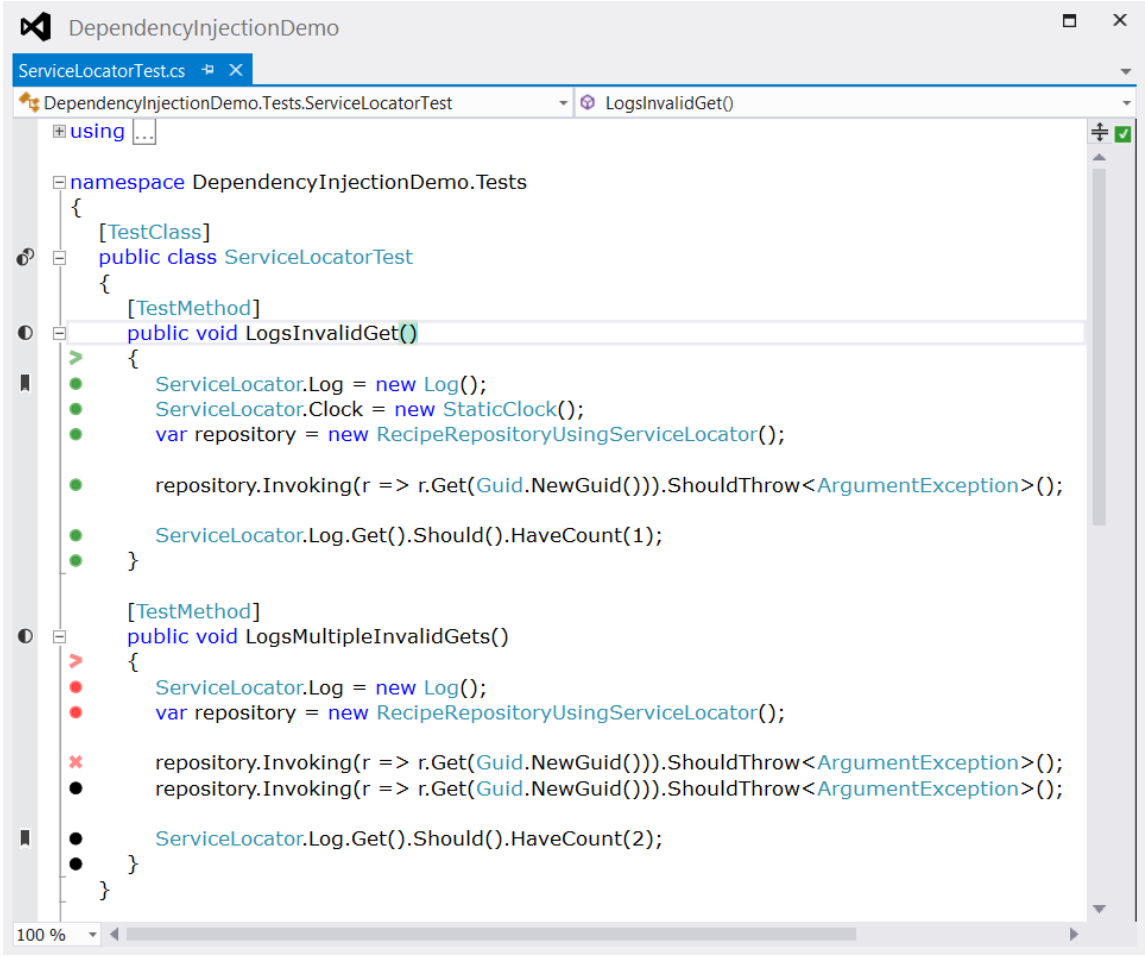
private Log, Clock

```
public class RecipeRepositoryUsingServiceLocator :
IRecipeRepository
{
    private readonly Dictionary<Guid, Recipe> _recipes =
      new Dictionary<Guid, Recipe>();
    private readonly ILog _log;
    private readonly IClock _clock;
...
}
```

Full run: passed



NCrunch: failed



NCrunch: failed

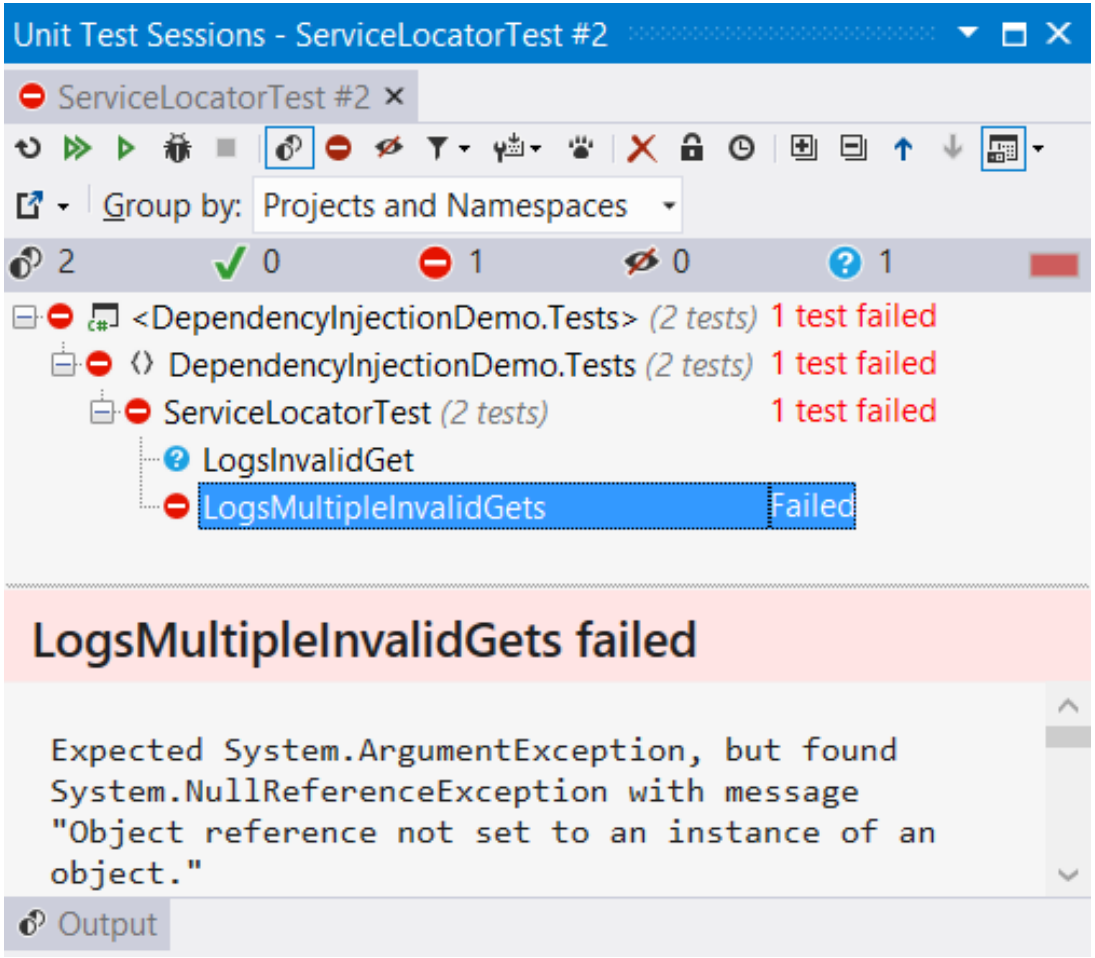
```
    [TestMethod]
    public void LogsMultipleInvalidGets()
    {
        ServiceLocator.Log = new Log();
        var repository = new RecipeRepositoryUsingServiceLocator();

        repository.Invoking(r => r.Get(Guid.NewGuid())).ShouldThrow
        repository.Invoking(r => r.Get(Guid.NewGuid())).ShouldThrow

        ServiceLocator.Log.Get().Should().HaveCount(2);
    }
}
```

Test: NullReferenceException

```
[TestMethod]
public void LogsMultipleInvalidGets()
{
    ServiceLocator.Log = new Log();
    var repository = new
        RecipeRepositoryUsingServiceLocator();
    repository.Invoking(r => r.Get(Guid.NewGuid()))
        .ShouldThrow<ArgumentException>();
```

```
Expected System.ArgumentException, but found
System.NullReferenceException with message
"Object reference not set to an instance of an
object."
```

isolierter Test: failed



Test: Initialisierung vergessen

```
[TestMethod]
public void LogsMultipleInvalidGets()
{
    ServiceLocator.Log = new Log();
    ServiceLocator.Clock = new StaticClock();
    var repository = new
        RecipeRepositoryUsingServiceLocator();
```

via Dependency Injection

```
public RecipeRepositoryUsingDependencyInjection
    (ILog log, IClock clock)
{
    _log = log;
    _clock = clock;
}
```

Test: explizite Parameter

```
[TestMethod]
public void LogsInvalidGet()
{
    var log = new Log();
    var clock = new StaticClock();
    var repository =
       new   RecipeRepositoryUsingDependencyInjection
       (log, clock);

    repository.Invoking(r => r.Get(Guid.NewGuid()))
       .ShouldThrow<ArgumentException>();

    log.Get().Should().HaveCount(1);
}
```

Test: exakte Zeit

```
[TestMethod]
public void LogsTimeOnInvalidGet()
{
    var log = new Log();
    var clock = new StaticClock();
    var repository =
       new RecipeRepositoryUsingDependencyInjection
       (log, clock);

    repository.Invoking(r => r.Get(Guid.NewGuid()))
       .ShouldThrow<ArgumentException>();

    log.Get().First().Time.Should().Be(clock.Now);
}
```

Test: Mock

```
var log = new Mock<ILog>();
log.Setup(l => l.Error(It.IsAny<DateTime>(),
    It.IsAny<string>()));
var clock = new Mock<IClock>();
var repository =
    new RecipeRepositoryUsingDependencyInjection
    (log.Object, clock.Object);

repository.Invoking(r => r.Get(Guid.NewGuid()))
    .ShouldThrow<ArgumentException>();

log.Verify(l => l.Error(It.IsAny<DateTime>(),
    It.IsAny<string>()), Times.Exactly(1));
```

moq

Test: Auto-Mock

```
[TestMethod]
public void ThrowsOnInvalidGet()
{
    var fixture = new Fixture()
        .Customize(new AutoMoqCustomization());
    var repository = fixture.Create
        <RecipeRepositoryUsingDependencyInjection>();

    repository.Invoking(r => r.Get(Guid.NewGuid()))
        .ShouldThrow<ArgumentException>();
}
```

AutoFixture

Inversion of Control

- gegen Interfaces programmieren
- Implementierung zuliefern lassen
- **Abhängigkeiten automatisch auflösen**

Ninject

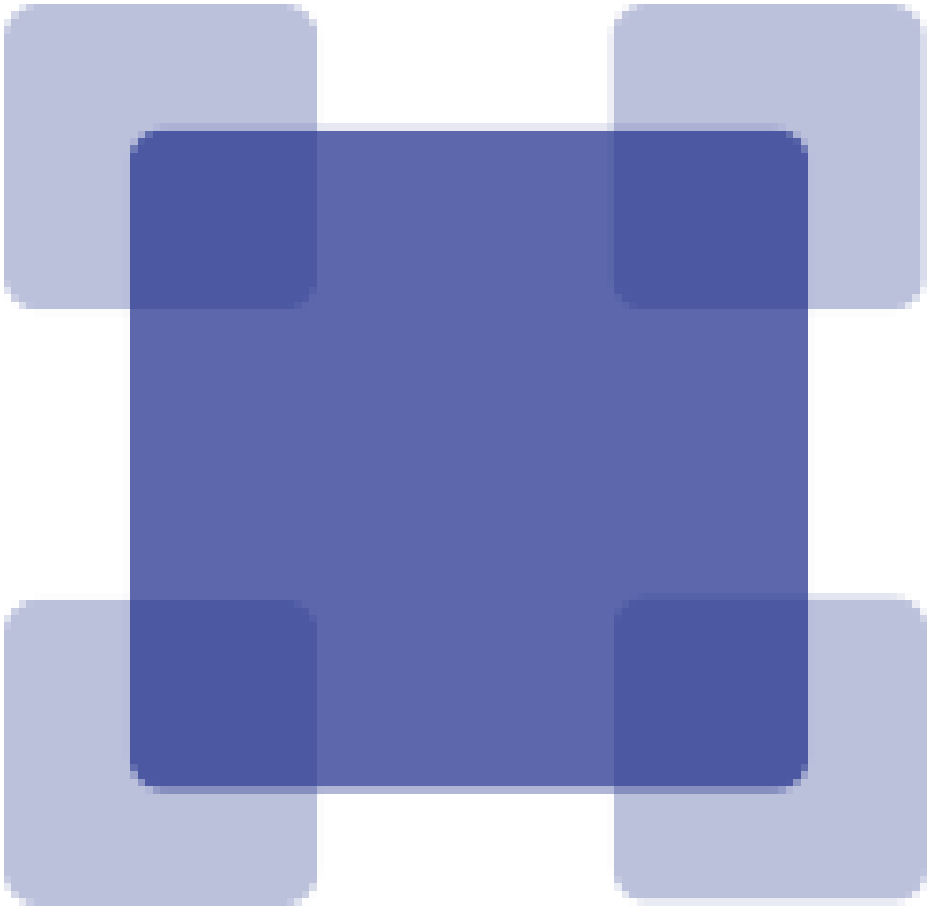
Test: Ninject

```
var kernel = new StandardKernel();
kernel.Bind<IClock>().To<StaticClock>();
kernel.Bind<ILog>().To<Log>().InSingletonScope();
kernel.Bind<IRecipeRepository>()
    .To<RecipeRepositoryUsingDependencyInjection>();

var log = kernel.Get<ILog>();
var repository = kernel.Get<IRecipeRepository>();

repository.Invoking(r => r.Get(Guid.NewGuid()))
    .ShouldThrow<ArgumentException>();

log.Get().Should().HaveCount(1);
```

Test: Ninject

```
var kernel = new StandardKernel();
kernel.Bind<IClock>().To<StaticClock>();
kernel.Bind<ILog>().To<Log>().InSingletonScope();
kernel.Bind<IRecipeRepository>()
    .To<RecipeRepositoryUsingDependencyInjection>();

var log = kernel.Get<ILog>();
var repository = kernel.Get<IRecipeRepository>();

repository.Invoking(r => r.Get(Guid.NewGuid()))
    .ShouldThrow<ArgumentException>();

log.Get().Should().HaveCount(1);
```

Castle Windsor



Test: Castle Windsor

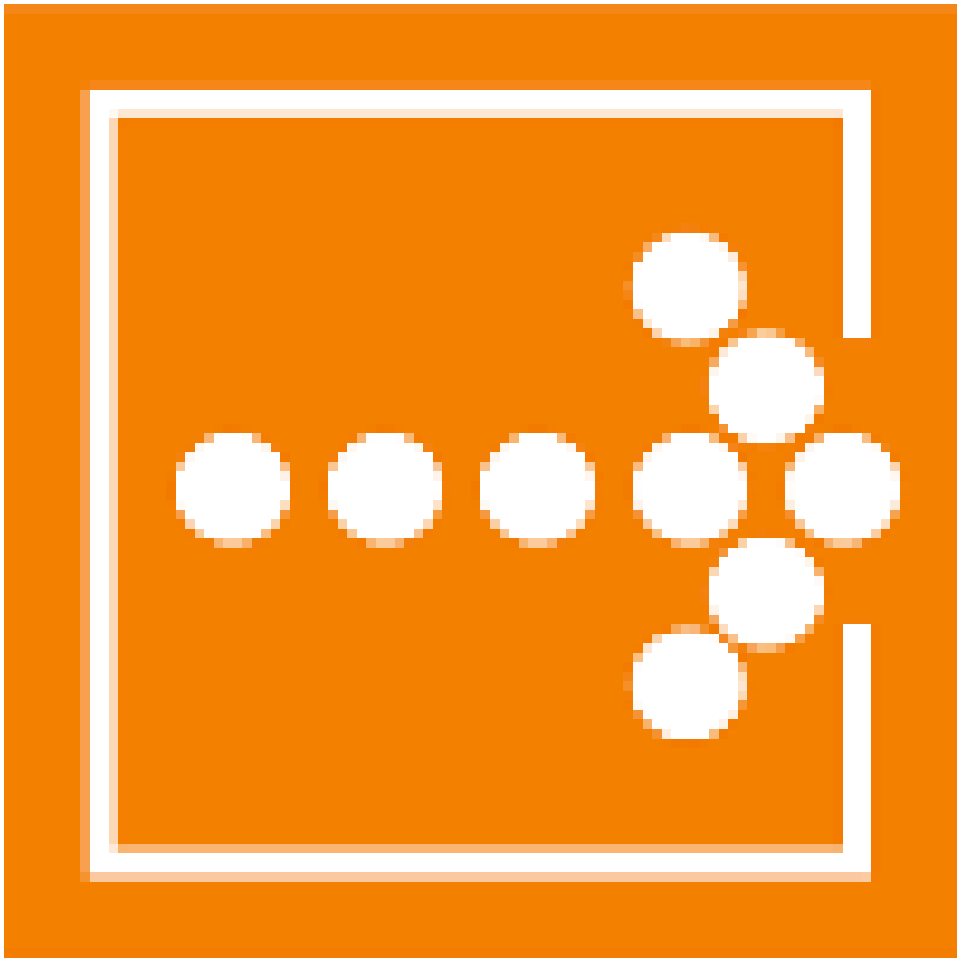
```
var container = new WindsorContainer();
container.Register(Component.For<ILog>()
    .ImplementedBy<Log>().LifestyleSingleton());
container.Register(Component.For<IClock>()
    .ImplementedBy<StaticClock>());
container.Register(Component.For<IRecipeRepository>()
    .ImplementedBy<RecipeRepositoryUsingDependencyInjection>().LifestyleTransient());

var log = container.Resolve<ILog>();
var repository = container.Resolve<IRecipeRepository>();
```

StructureMap

Test: StructureMap

```
var container = new Container(x =>
{
    x.For<ILog>().Singleton().Use<Log>();
    x.For<IClock>().Use<StaticClock>();
    x.For<IRecipeRepository>().
      Use<RecipeRepositoryUsingDependencyInjection>();
});

var log = container.GetInstance<ILog>();
var repository =
   container.GetInstance<IRecipeRepository>();
```

Unity



Test: Unity

```
var container = new UnityContainer();
container
    .RegisterType<ILog, Log>(
        new ContainerControlledLifetimeManager())
    .RegisterType<IClock, StaticClock>()
    .RegisterType<IRecipeRepository,
        RecipeRepositoryUsingDependencyInjection>();

var log = container.Resolve<ILog>();
var repository = container.Resolve<IRecipeRepository>();
```

Kernel als Parameter

Nicht den Kernel als Parameter übergeben, Kernel ist keine Factory!

Console Application

```
class RecipeCounter
{
    private readonly SystemClock _clock;
    private readonly
    RecipeRepositoryUsingDependencyInjection _repository;

    public RecipeCounter(SystemClock clock,
        RecipeRepositoryUsingDependencyInjection
        repository)
    {
        _clock = clock;
        _repository = repository;
    }
```

Console Application

```
    public void Write()
    {
        System.Console.WriteLine(
            "There are {0} recipes available at {1}.",
            _repository.Count,
            _clock.Now.ToLongTimeString());
    }
}
```

Console Application

```
static void Main(string[] args)
{
    var kernel = new StandardKernel();
    kernel.Bind<IClock>().To<SystemClock>();
    kernel.Bind<ILog>().To<Log>();
    kernel.Bind<IRecipeRepository>().
        To<RecipeRepositoryUsingDependencyInjection>();

    var recipeCounter = kernel.Get<RecipeCounter>();
    recipeCounter.Write();
}
```

ASP.NET MVC

```
[assembly:
WebActivator.PreApplicationStartMethod(typeof(Dependen
cyInjectionDemo.WebMvc.App_Start.NinjectWebCommon),
"Start")]
[assembly:
WebActivator.ApplicationShutdownMethodAttribute(typeof(
DependencyInjectionDemo.WebMvc.App_Start.NinjectWeb
Common), "Stop")]

public static class NinjectWebCommon
{
    ...
}
```

ASP.NET MVC

```
public static void Start()
{
    DynamicModuleUtility.RegisterModule(
        typeof(OnePerRequestHttpModule));
    DynamicModuleUtility.RegisterModule(
        typeof(NinjectHttpModule));
    bootstrapper.Initialize(CreateKernel);
}

public static void Stop()
{
    bootstrapper.ShutDown();
}
```

ASP.NET MVC

```
private static IKernel CreateKernel()
{
    var kernel = new StandardKernel();
    kernel.Bind<Func<IKernel>>().ToMethod(
        ctx => () => new Bootstrapper().Kernel);
    kernel.Bind<IHttpModule>()
        .To<HttpApplicationInitializationHttpModule>();

    RegisterServices(kernel);
    return kernel;
}
```

ASP.NET MVC

```
private static void RegisterServices(IKernel kernel)
{
    kernel.Bind<IClock>().To<SystemClock>();
    kernel.Bind<ILog>().To<Log>();
    kernel.Bind<IRecipeRepository>().
        To<RecipeRepositoryUsingDependencyInjection>();
}
```

ASP.NET MVC: Controller

```
public HomeController(IClock clock,
    IRecipeRepository repository)
{
    _clock = clock;
    _repository = repository;
}
```

SharePoint

„The SharePoint Service Locator is a reusable component that provides a simple implementation of the service locator pattern.”

[MSDN, Application Foundations for SharePoint 2010]

➔ Anti-Pattern

Konstruktor vs. Property

```
private class SelfInjectingClass
{
    [Inject]
    public ILog Log { get; set; }

    public SelfInjectingClass()
    {
        KernelLocator.Kernel.Inject(this);
    }

    public SelfInjectingClass(ILog log)
    {
        Log = log;
    }
}
```

Inject This

```
[TestMethod]
public void InjectThis()
{
    var kernel = new StandardKernel();
    KernelLocator.Kernel = kernel;
    kernel.Bind<ILog>().To<Log>().InSingletonScope();

    var selfInjectingClass = new SelfInjectingClass();

    selfInjectingClass.Log.Should().NotBeNull();
}
```

Inject

```
[TestMethod]
public void Inject()
{
    var kernel = new StandardKernel();
    kernel.Bind<ILog>().To<Log>().InSingletonScope();

    var selfInjectingClass =
        kernel.Get<SelfInjectingClass>();

    selfInjectingClass.Log.Should().NotBeNull();
}
```

Zusammenfassung

- Abhängigkeiten automatisch auflösen
- Lebenzyklen beachten
- Work-Around für Frameworks möglich

- Folien, Code
  http://malteclasen.de/blog
- Beratung, Training
  info@malteclasen.de