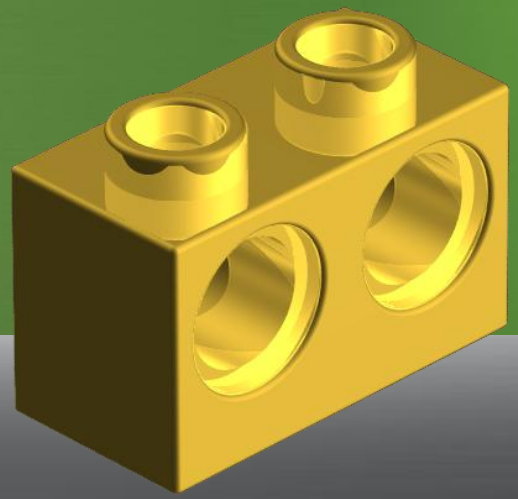
Lego for Engineers

- How to build Reusable and Maintainable Applications in C#

Theo Jungeblut 10/10/2010



Now that's Smart.

Omnicell®

Theo Jungeblut

- Senior Software Developer at Omnicell Inc. in Mountain View
- Designs and implements .NET based applications for more than 6 ½ years
- Previously was working for 3 ½ years in factory automation with focus on component based software and framework development
- Degree in Software Engineering and Network Communications

theo.jungeblut@gmail.com

Overview

- Why Lego for Software Engineers?
- “Keep It Simple Stupid”-Principle (KISS)
  - The Power of Simplicity
  - Two Ways of doing Something Similar
- Design Patterns and Principles
- Dependency Injection Container & more
- Summary
- References
- Q & A

Why Lego for Software Engineers?

Lego (trademarked in capitals as LEGO) is a line of construction toys manufactured by the Lego Group

http://upload.wikimedia.org/wikipedia/commons/7/75/Lego_technic_gears.jpg

KISS-Principle – “Keep It Simple Stupid”

by Kelly Johnson

http://blogs.smarter.com/blogs/Lego%20Brick.jpg

http://blog.makezine.com/intro.jpg

“Keep It Simple Stupid” design principal by Kelly Johnson

The Power of Simplicity

http://www.sharenator.com/Lego_Art/05_lego_art-2748.html

http://www.bitrebels.com/geek/cant-afford-a-car-build-a-lego-one/

http://www.geekalerts.com/lego-iphone/

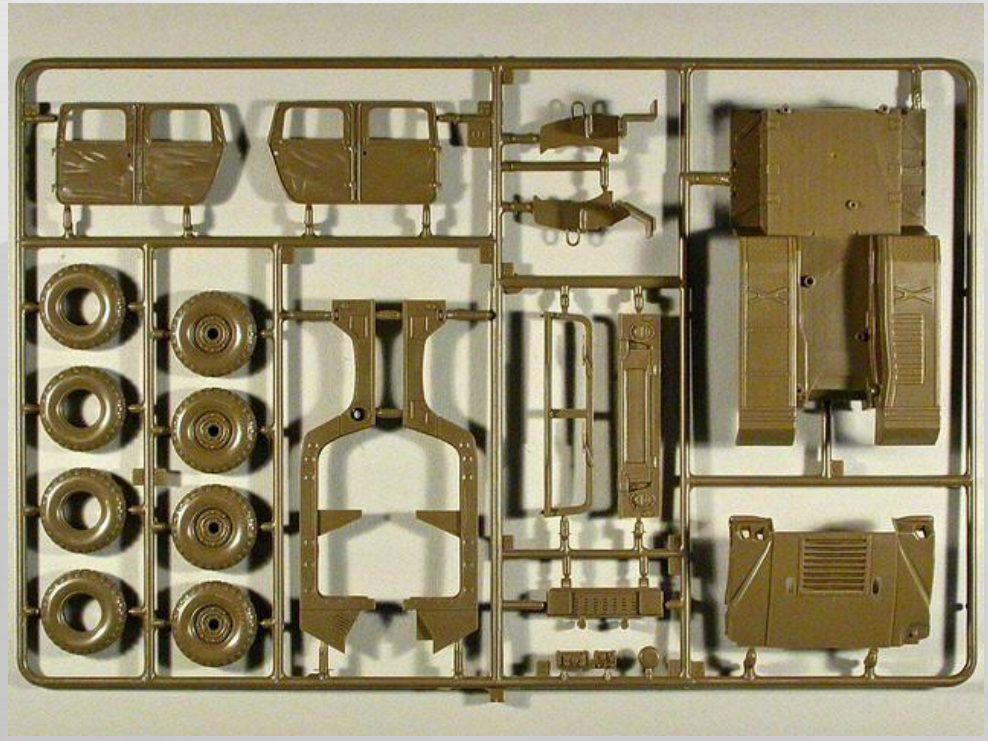
Different Ways of doing Something Similar

http://www.ericalbrecht.com

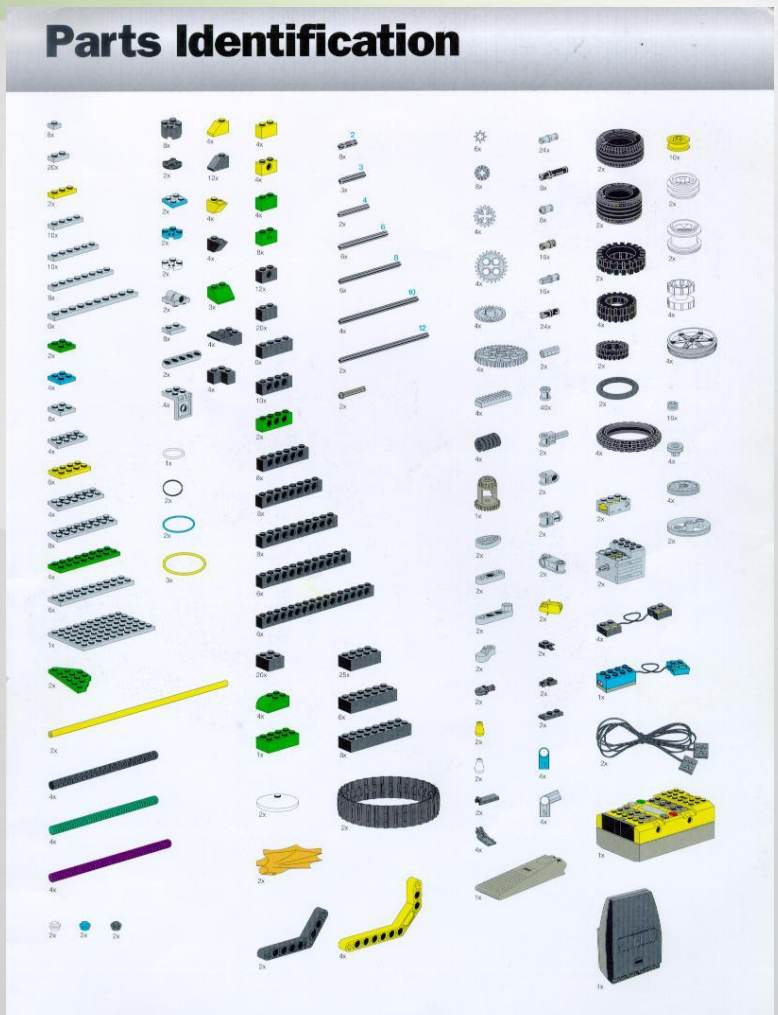
http://www.ericalbrecht.com

http://www.julianaheng.com/transformers-rotf-bumblebee-and-sam-action-figures/

Why Reusable Components Rock



http://www.modellversium.de/kit/artikel.php?id=1922



http://www.wilcoxusa.net/mindstorms/images/constructopedia/cs10-47-parts_identification.jpg

Why Reusable Components Rock

http://www.ericalbrecht.com/technic/8020/8020all.jpg

Design Patterns and Principals

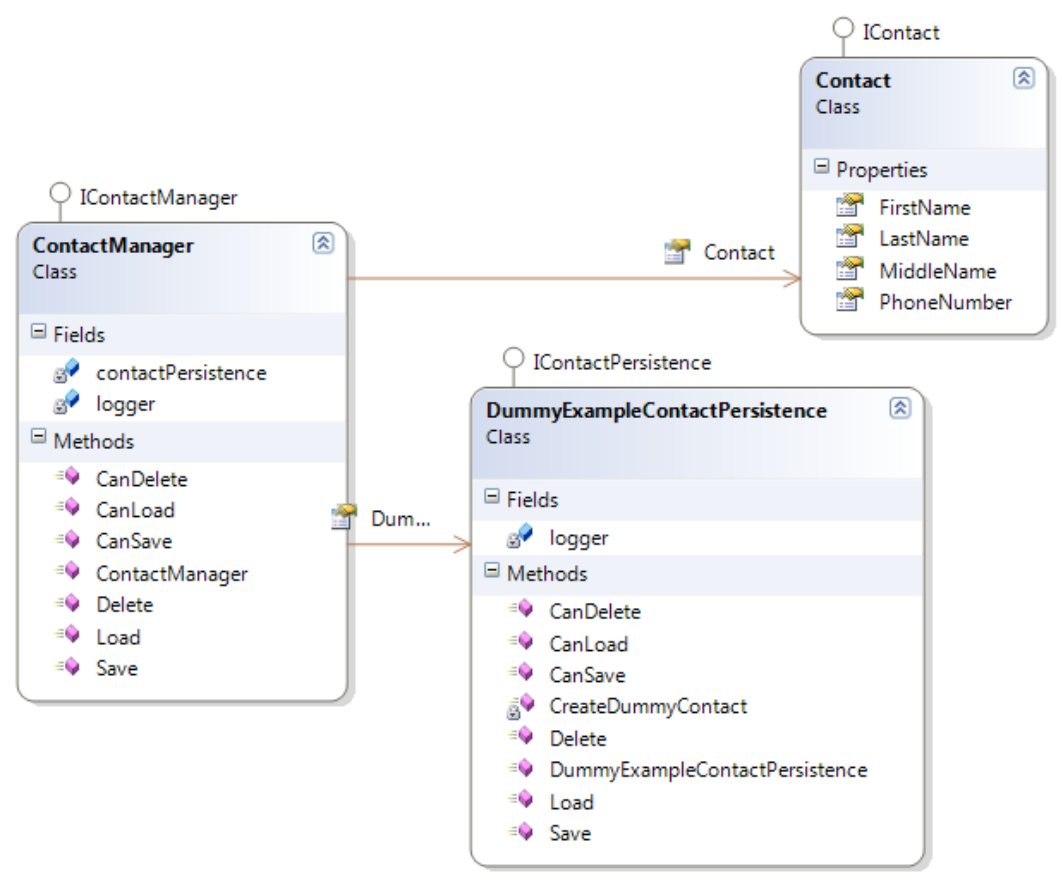
- Separation of Concerns (SoC)
- Single Responsibility Principle (SRP)
- Component Oriented Programming (CoP)
- Interface / Contract
- Don’t Repeat Yourself (DRY)
- You Ain't Gonna Need It (YAGNI)
- Inversion of Control (IoC)
    - Constructor Injection
    - Setter Injection
    - Interface Injection
    - Service Locator

Separation of Concerns (SoC)

probably by Edsger W. Dijkstra in 1974

• In computer science, separation of concerns (SoC) is the process of separating a computer program into distinct features that overlap in functionality as little as possible.

•A concern is any piece of interest or focus in a program. Typically, concerns are synonymous with features or behaviors.



http://en.wikipedia.org/wiki/Separation_of_Concerns

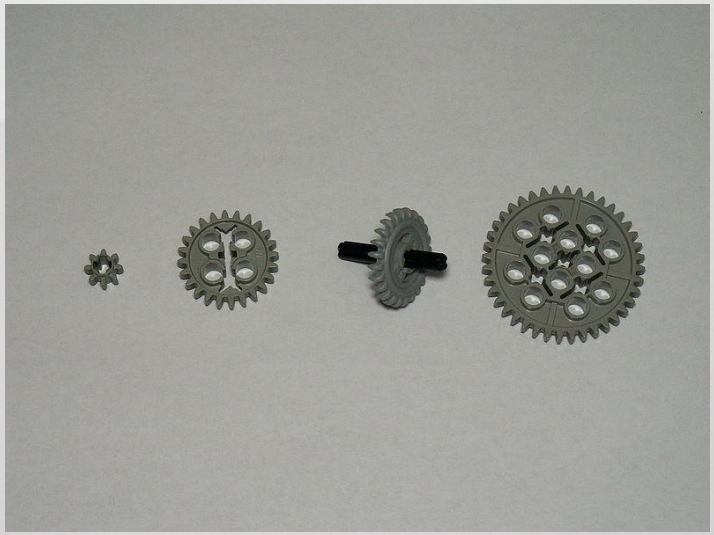
Single Responsibility Principle(SRP)

by Robert C Martin

Every object should have a single responsibility, and that responsibility should be entirely encapsulated by the class.

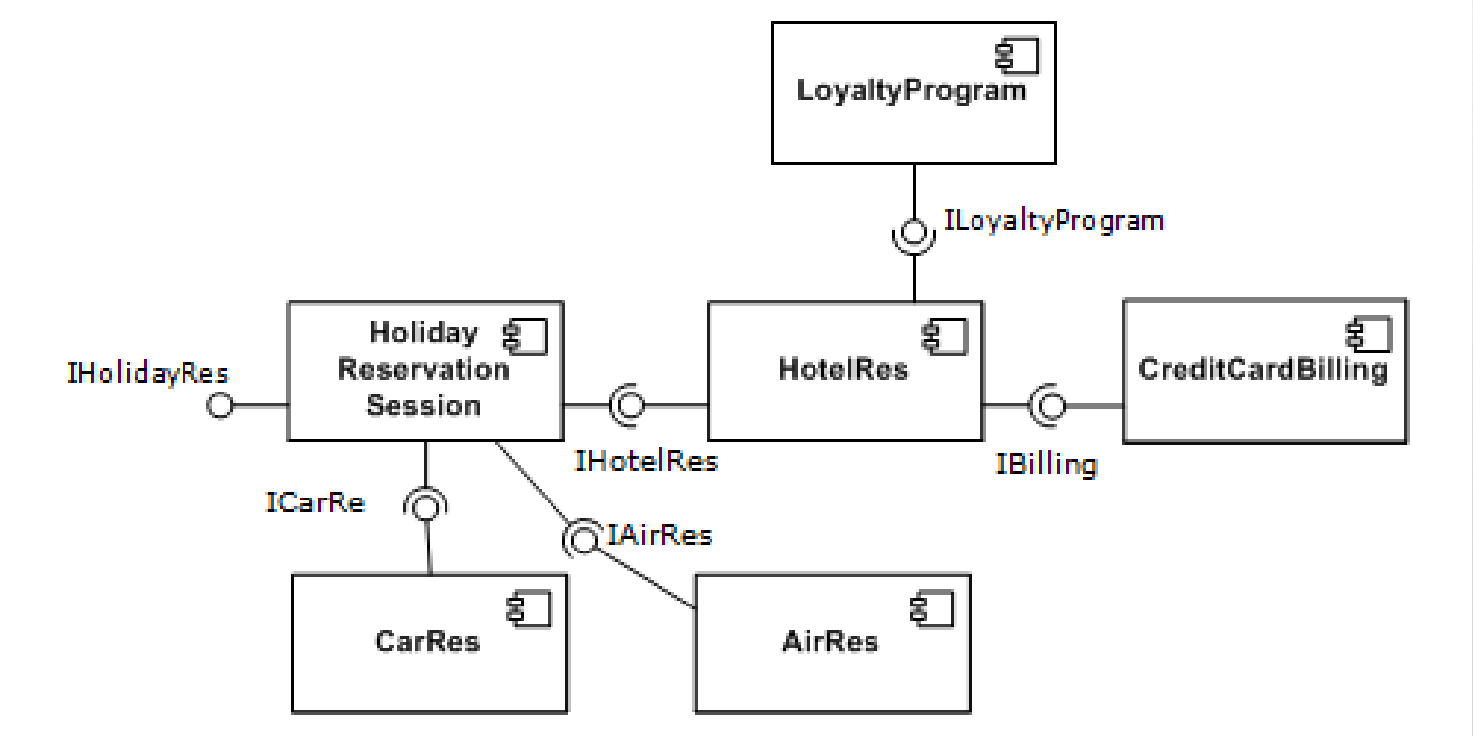
http://en.wikipedia.org/wiki/Single_responsibility_principle

```
public class Timer : IDisposable
   {
      public event
EventHandler<ElapsedEventArgs> Elapsed;

      public int IntervalInMilliseconds { get; set; }

      public bool Enabled { get; }

      public void Start(){…};
      public void Stop(){..};
   }
```



http://www.ericalbrecht.com

Component Oriented Programming (CoP)



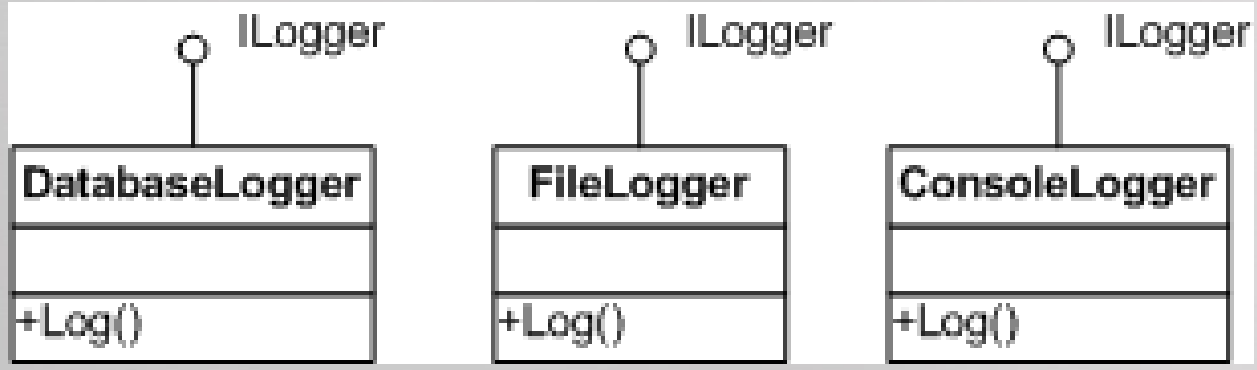
http://upload.wikimedia.org/wikipedia/en/2/25/Component-based_Software_Engineering_(CBSE)_-_example_2.gif

Interfaces / Contracts

- Decouple Usage and Implementation through introduction of contract
- Allows to replace implementation without changing the consumer

```
public interface ILogger
{
    void Log(Message message);
}
```

```
public class LoggingTest
{
    void Test Logging(ILogger logger)
    {
        logger.Log(new Message("Hallo");
    }
}
```



Don’t Repeat Yourself (DRY)

by Andy Hunt and Dave Thomas in their book “The Pragmatic Programmer”

```
// Code Copy and Paste Method
public Class Person
 {
   public string FirstName { get;  set;}
   public string LastName { get; set;}

   public Person(Person person)
   {
      this.FirstName = string.IsNullOrEmpty(person.FirstName)
                  ? string.Empty : (string) person.FirstName.Clone();

      this.LastName = string.IsNullOrEmpty(person.LastName)
                  ? string.Empty : (string) person.LastName.Clone();
   }

   public object Clone()
   {
      return new Person(this);
   }
}
```

```
// DRY Method
public Class Person
 {
   public string FirstName { get;  set;}
   public string LastName { get; set;}

   public Person(Person person)
   {
      this.FirstName = person.FirstName.CloneSecured();
      this.LastName = person.LastName.CloneSecured();
   }

   public object Clone()
   {
      return new Person(this);
   }
}
```

```
public static class StringExtension
{
   public static string CloneSecured(this string original)
   {
     return string.IsNullOrEmpty(original) ? string.Empty : (string)original.Clone();
   }
}
```

You Ain't Gonna Need It (YAGNI)

by Ron E. Jeffries

What to avoid:

- Gold Plating
- Feature Creep
- Code Blow

Because new feature need to be:

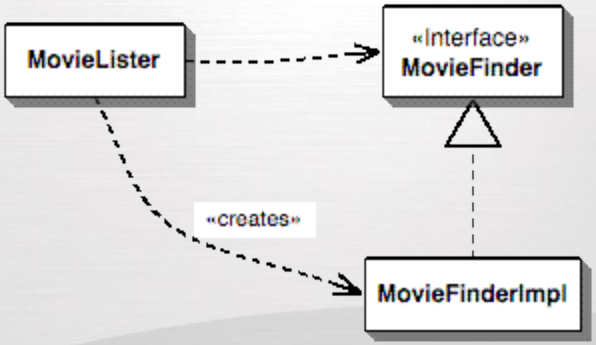
- Implemented
- Tested
- Documented
- Maintained

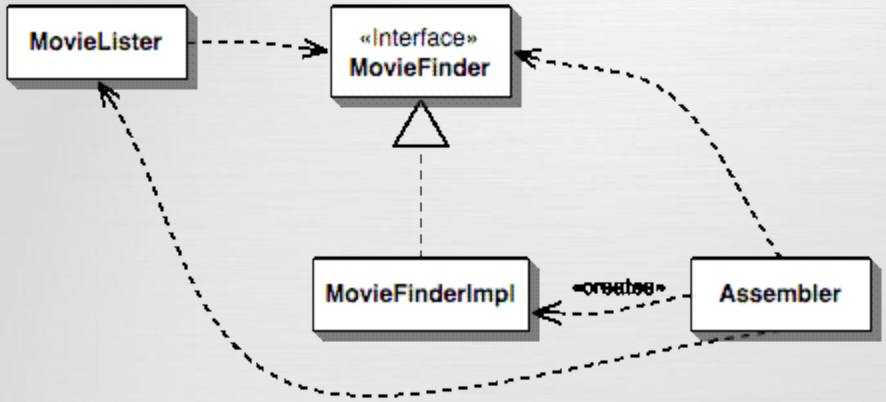
Balance concerns

- Implement only what is required but design as far as needed

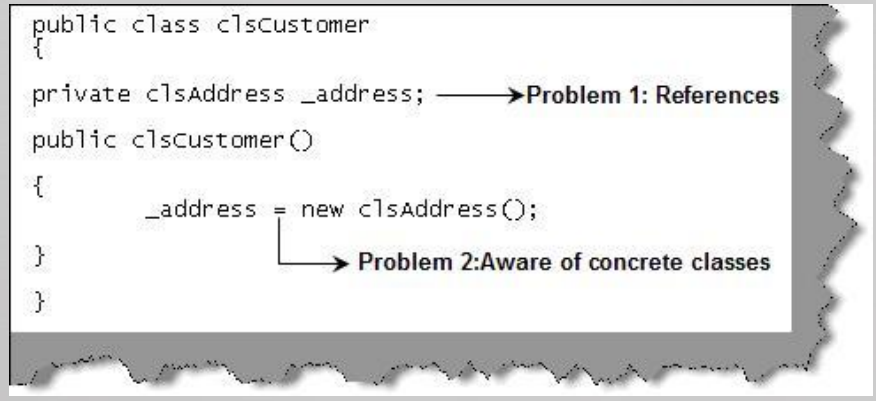
Inversion of Control (IoC)

by Martin Fowler 1994

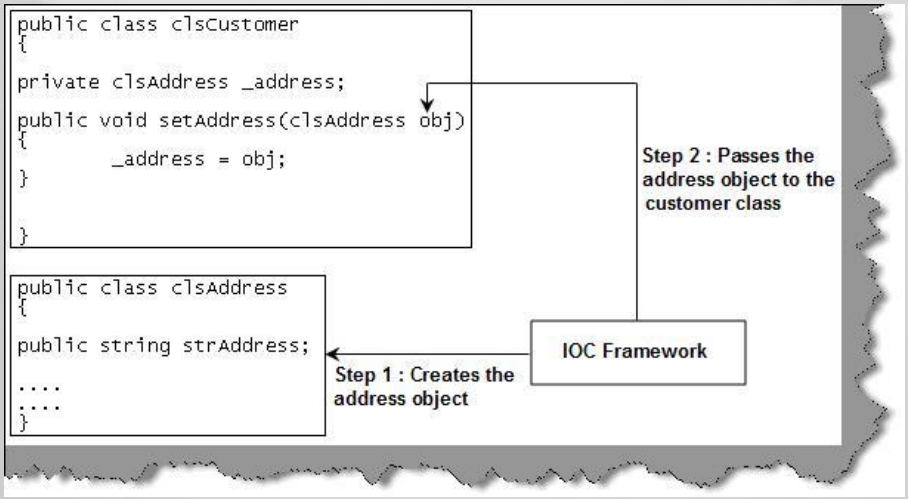




http://www.martinfowler.com/articles/injection.html



http://www.codeproject.com/KB/aspnet/IOCDI/ProblemsofIOC.JPG



http://www.codeproject.com/KB/aspnet/IOCDI/IOCframework.JPG

Inversion of Control (IoC) - Constructor Injection

http://www.martinfowler.com/articles/injection.html

```
// UNITY Example
public class CustomerService
{
    public CustomerService(LoggingService myServiceInstance)
    {
        // work with the dependent instance
        myServiceInstance.WriteToLog("SomeValue");
    }
}
```

http://msdn.microsoft.com/en-us/library/ff650320.aspx

Inversion of Control (IoC) – Setter (Property) Injection

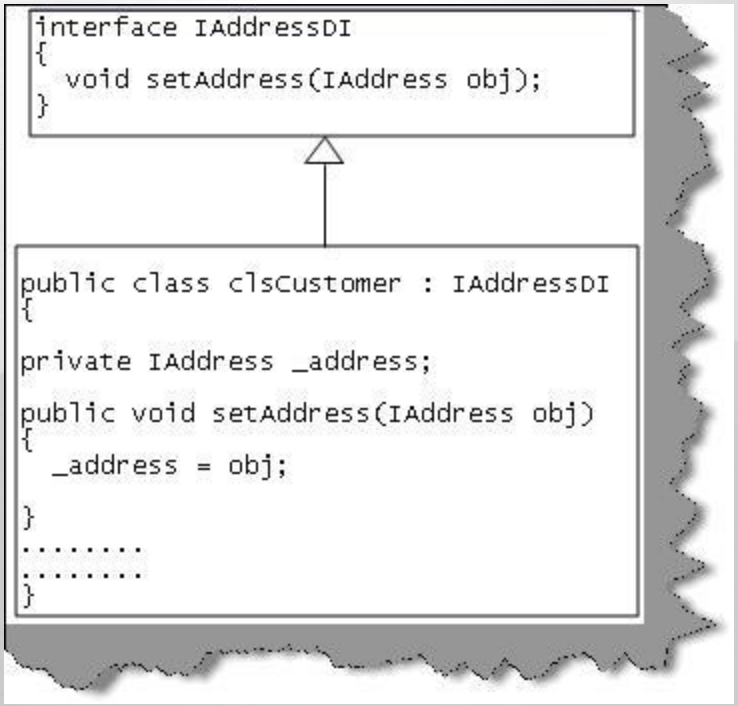
http://www.martinfowler.com/articles/injection.html

```
// UNITY Example
public class ProductService
{
  private SupplierData supplier;

  [Dependency]
  public SupplierData SupplierDetails
  {
    get { return supplier; }
    set { supplier = value; }
  }
}
```

http://msdn.microsoft.com/en-us/library/ff650320.aspx

Inversion of Control (IoC) – Interface Injection

http://www.martinfowler.com/articles/injection.html

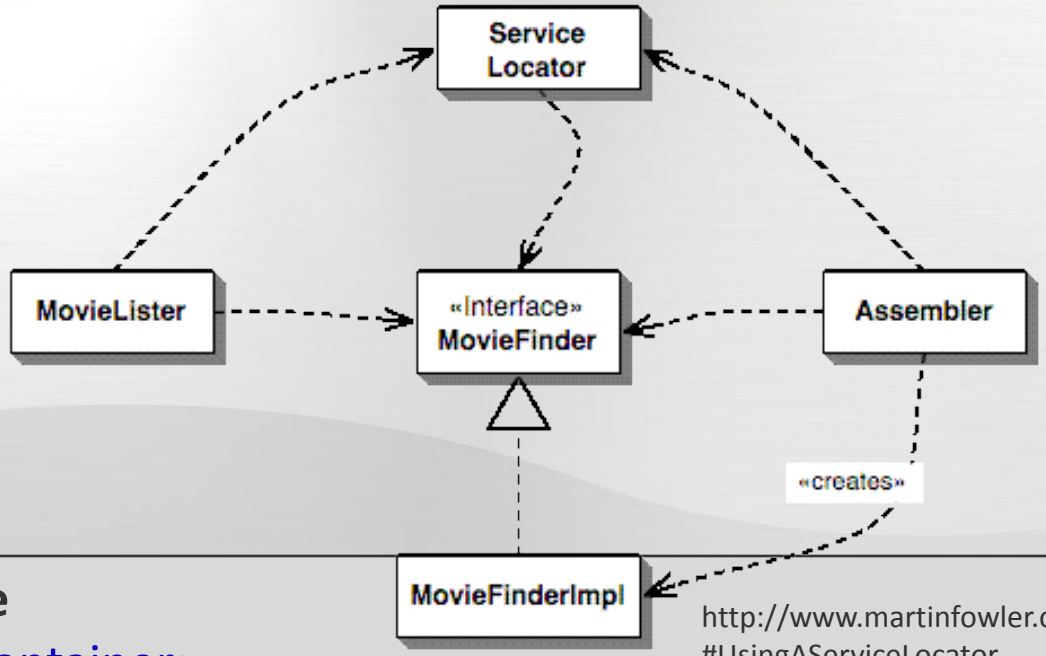


In this methodology we implement an interface from the IOC framework. IOC framework will use the interface method to inject the object in the main class. You can see in figure 'Interface based DI' we have implemented an interface 'IAddressDI' which has a 'setAddress' method which sets the address object. This interface is then implemented in the customer class. External client / containers can then use the 'setAddress' method to inject the address object in the customer object.

http://www.codeproject.com/KB/aspnet/IOCDI/InterfacebasedDI.JPG http://www.codeproject.com/KB/aspnet/IOCDI.aspx

Inversion of Control (IoC) – Service Locator

http://www.martinfowler.com/articles/injection.htm
l



// UNITY Example

```
IUnityContainer container;
container = new UnityContainer();

container.RegisterType<IMyDummyService, StupidDummyService>();

IMyDummyService myServiceInstance = container.Resolve<IMyDummyService>();
```

http://www.martinfowler.com/articles/injection.html
#UsingAServiceLocator

Inversion of Control - Service Locator vs Dependency Injection

http://www.martinfowler.com/articles/injection.html#ServiceLocatorVsDependencyInjection

- Service Locator allows to request explicitly the needed instance/type/service

- Every user of a service has a dependency to the Service Locator
  - Potential issue if the component need to be provided to 3rd parties.
  - Favorable for closed platforms as the Service Locator allow more control

- Testing is easier with Dependency Injection than a Service Locator if Service provided is not easily substituted

- In general Service Locator is only the less compelling choice if the code is mainly used out of the control of the writer

Dependency Injection Container & more

- **Typically support all types of Inversion of Control mechanisms**
  - Constructor Injection
  - Property (Setter) Injection
  - Interface Injection
  - Service Locator

- **.NET based DI-Container**
  - Unity
  - Castle Windsor
  - StructureMap
  - Spring.NET
  - Autofac
  - Puzzle.Nfactory
  - Ninject
  - PicoContainer.NET
  - and more

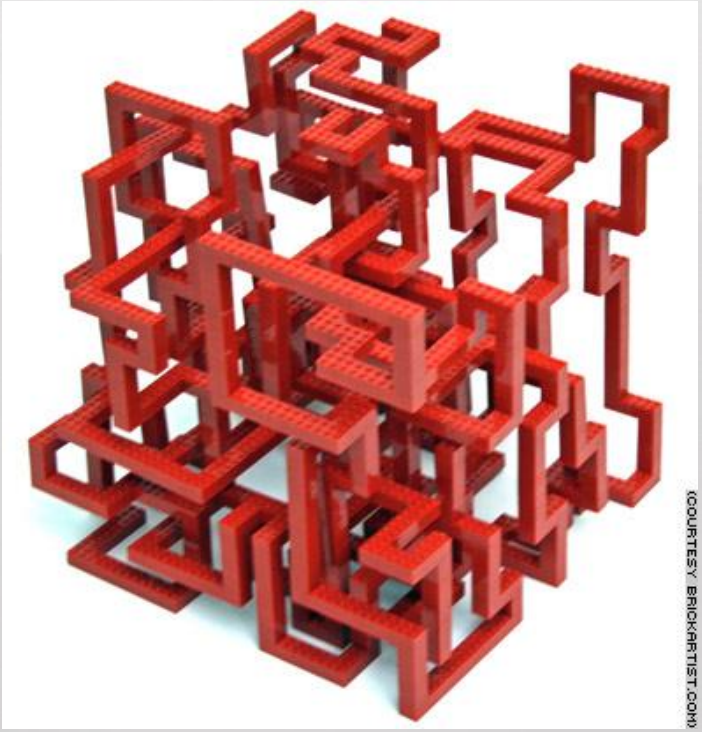
Related Technology:

- Managed Extensibility Framework (MEF)
- Windows Communication Foundation (WCF)

Summary

Improving Reusability and Maintainability through:

- Simplification and Specialization
  (KISS, SoC, SRP)
- Decoupling
  (Interface, CoP, IoC or SOA)
- Avoiding Code Blow (DRY, YAGNI)
- Testability (all of them!)



http://files.sharenator.com/11_lego_art_Lego_Art-s396x414-2755.jpg

References

http://en.wikipedia.org/wiki/Keep_it_simple_stupid
http://picocontainer.org/patterns.html
http://en.wikipedia.org/wiki/Separation_of_concerns
http://en.wikipedia.org/wiki/Don't_repeat_yourself
http://en.wikipedia.org/wiki/You_ain't_gonna_need_it
http://en.wikipedia.org/wiki/Component-oriented_programming
http://en.wikipedia.org/wiki/Service-oriented_architecture
http://www.martinfowler.com/articles/injection.html
http://www.codeproject.com/KB/aspnet/IOCDI.aspx
http://msdn.microsoft.com/en-us/magazine/cc163739.aspx
http://msdn.microsoft.com/en-us/library/ff650320.aspx
http://msdn.microsoft.com/en-us/library/aa973811.aspx
http://msdn.microsoft.com/en-us/library/ff647976.aspx
http://msdn.microsoft.com/en-us/library/cc707845.aspx
http://msdn.microsoft.com/en-us/library/bb833022.aspx
http://dotnetslackers.com/articles/net/A-First-Look-at-Unity-2-0.aspx
http://unity.codeplex.com/
http://www.idesign.net/idesign/DesktopDefault.aspx?tabindex=5&tabid=11

Q & A

http://www.sharenator.com/Lego_Art/03_lego_art-2746.html