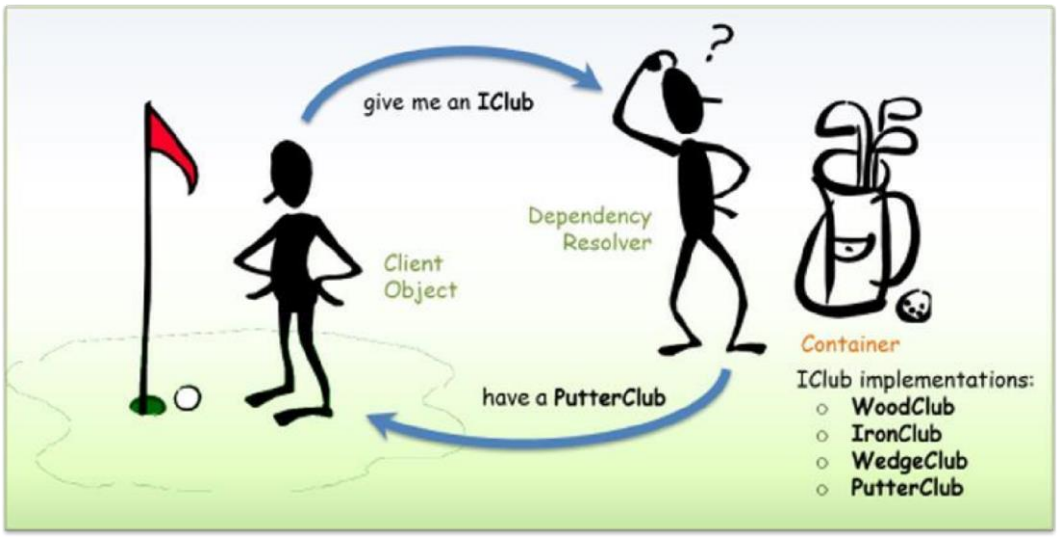
Av **Erik ”Zkall” Svensson**
@esdev.se

Dependency Injection (IoC)



Vad är dependency injection?

Dependency injection är som att skriva var apa = new Apa(); Men att du skippar new utan istället injectar det från en container. Man kan säga att du återanvänder begagngade klasser med hjälp av dependency injection, så att du slipper använda nya objekt hela tiden.

http://en.wikipedia.org/wiki/Dependency_injection är en bra artikel om dependency injection, läs den så går jag inte för djupt på det.

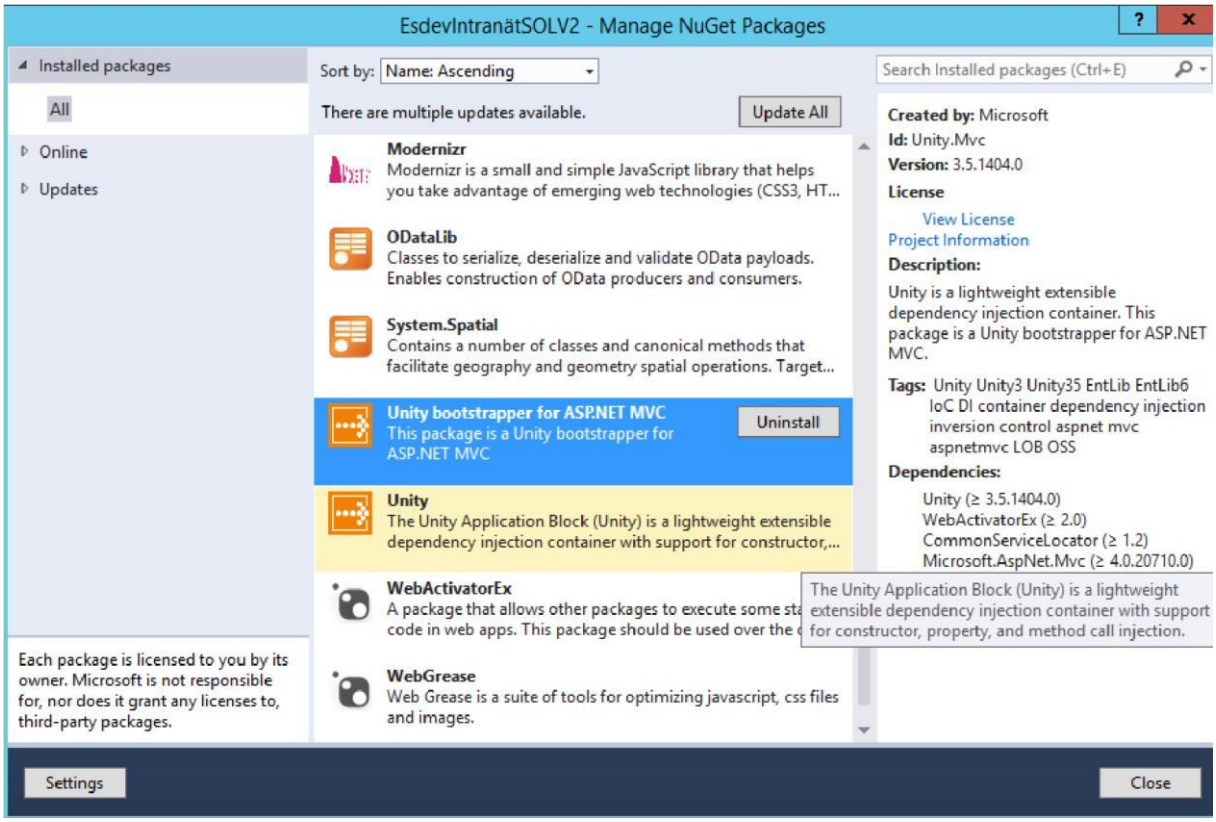
Hur ser dependency injection ut i kod?

```
public class AdminController : Controller
{
    private IUsersBusinessLogicLayer BLL;
    private IForumBusinessLogicLayer F_BLL;
    0 references | Erik Zkall. Svensson | 1 change
    public AdminController(IUsersBusinessLogicLayer _BLL, IForumBusinessLogicLayer _forum)
    {
        BLL = _BLL;
        F_BLL = _forum;

    }
    // GET: Admin
    0 references | Erik Zkall. Svensson | 4 changes
```

Du initierar först det interfacet du ska implementa för att få en klass utan värde. Sedan i konstruktorn

(koden som kallas när controllern initieras.) Så skriver du som ovan. På det sättet injectar du en dependency till din controller och assignar den till din tomma variabel. Sedan är det bara att kalla på frid o fröjd!

Hur lägger man till en UnityContainer?



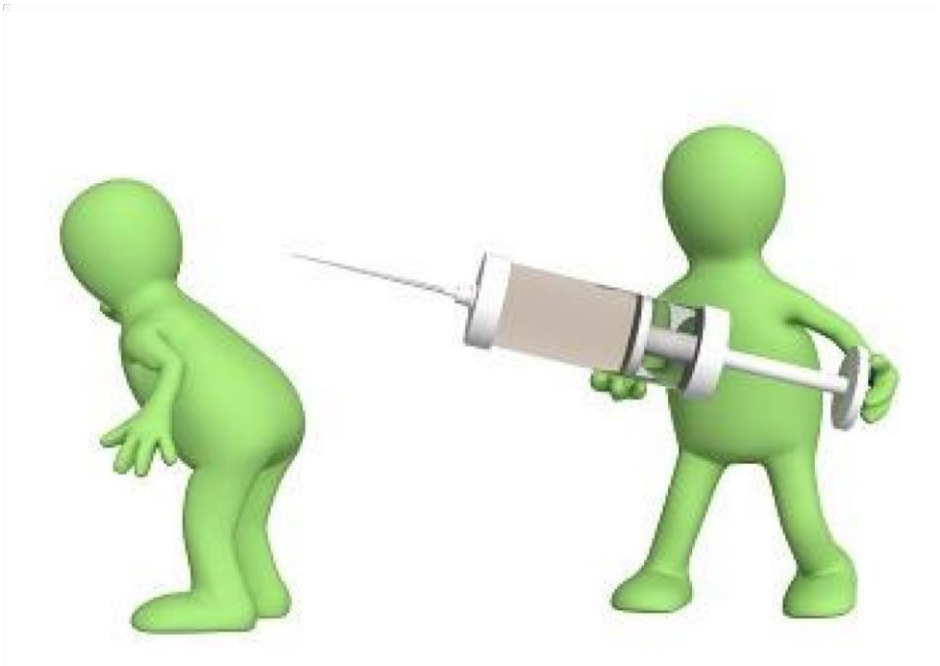
Högerklicka på projektet, välj manage nugget packages och sök sedan online efter unity, installera Unity bootstraper for MVC och Unity. Efter det är gjort har du fått 2 nya filer i App-start mappen. Endast en av dom är relevanta för användandet av dependency injection.

```
namespace EsdevIntranätSOLV2.App_Start
{
    /// <summary>
    /// Specifies the Unity configuration for the main container.
    /// </summary>
    2 references | Erik Zkall. Svensson | 5 changes
    public class UnityConfig
    {
        Unity Container

        /// <summary>Registers the type mappings with the Unity container.</summary>
        /// <param name="container">The unity container to configure.</param>
        /// <remarks>There is no need to register concrete types such as controllers or API controllers (unles
        /// change the defaults), as Unity allows resolving a concrete type even if it was not previously regi
        1 reference | Erik Zkall. Svensson | 5 changes
        public static void RegisterTypes(IUnityContainer container)
        {
            // NOTE: To load from web.config uncomment the line below. Make sure to add a Microsoft.Practices.
            // container.LoadConfiguration();

            // TODO: Register your types here
            container.RegisterType<ITFSBusinessLogicLayer, TFSBusinessLogicLayer>();
            container.RegisterType<IChattBusinessLogicLayer, ChattBusinessLogicLayer>();
            container.RegisterType<IForumBusinessLogicLayer, ForumBusinessLogicLayer>();
            container.RegisterType<IUsersBusinessLogicLayer, UsersBusinessLogicLayer>();
            container.RegisterType<IDokumentBusinessLogicLayer, DokumentBusinessLogicLayer>();
            container.RegisterType<IMenyBusinessLogicLayer, MenyBusinessLogicLayer>();
        }
    }
}
```

Öppna upp UnityConfig. Som ni ser i bilden ovan är det bara att kommentera ut koden med ctrl+k+u. Sedan skriver du ditt interface först, följt av classen som implementeras av den. Klart! Nu kan du refferera och injecta dina dependencys!

Men vänta nu, hur skapar man ett interface

Enkelt, du skapar classen och skriver klart den, när det är klart högerklickar du på classen, väljer refactor -> Extract -> Extract interface. Så skapar den sedan en fil med alla parametrar du väljer att skicka med som heter Iclass. Jag brukar bara trycka på add all public och sedan skapa interfacet.



Avslut

Varför dependency injection är bra:

1. Bättre för testing.
2. Återanvändbart
3. Användaren behöver inte kunna så mycket om koden som interfacet implementerar, användaren behöver bara veta att en metod gör x grej.
4. 2 programmerare kan jobba på 2 olika klasser som är beroende av varandra.
5. Dependency injection minskar hoppkoppling ´mellan klasser och dess beroenden