# [Dependency Injection](http://www.vasil.me/?p=215" \o "Permanent Link to Dependency Injection)

Луд умора няма. Вчера от нямане какво да правя реших – абе дай да си направя един простичък Dependency Resolver. И преди да опиша какво е това, мисля, че е по-добре да разясня какво е “обръщане на контрола” (Inversion of Control – IoC) и “инжектиране на зависимости” (dependency injection).

## **Inversion of Control**

Когато пишем проектите си, използвайки някой обектно-ориентиран език много често обичаме да пишем “конкретно”, а не “абстрактно”. Тоест обичаме да зависим на конкретни имплементации, а не на абстракции. Пример за това е, ако да речем имаме един клас **Lamp(лампа)** и един бутон за нейното включване – class **Button**. По-неопитните с ООП биха предложили да направим директна връзка между между лампата и бутона, като по този веднага бихме създали една зависимост. За да оперира коректно нашата лампа тя зависи от един бутон, чиято единствена цел би била да я включи. И това определено не е добра идея. Вместо можем да си изнесем един интерфейс (да го кръстим ILightSwitcher, например), чиято единствена цел би била да се разпорежда с включването на лампата. По този начин разкачваме двата класа и премахваме зависимостта между двата класа – и двата ще зависят от абстракция, а не от конкретни имплементация. Това е което наричаме обръщане на контрола – когато нашите модули от високо ниво (какъвто е лампата ни), не зависят от модули от по-ниско ниво (какъвто е нашият бутон). И двете зависят от абстракции.

## **Dependency Injection**

Какво представлява “инжектирането на зависимости” – това представлява шаблон за дизайн, който ни позволява да остраним здравите връзки между класовете и да ги подменяме по време на компилация или по дори и по време на изпълнението на нашата програма. При инжектирането на зависимости се открояват три елемента:

* Зависим елемент
* Декларация на зависимости – различни типове договори (интерфейси, абстрактни класове)
* Контейнер(container/resolver) на зависимостите

Какво ще рече това? Главната мотивация за създаването на този шаблон е да могат да се променят конкретните имплементации, които се използват по време на изпълнение на програмата, а не при компилиране. Това позволява кодът, който пишем да е много по модуларен и разкачен, което го прави в същото време и много по-тестваем.

## **Примерна имплементация на Dependency Resolver**

Има много различни IoC контейнери (например Ninject, Castle Windsor, Autofac за C# или Pico, Guice за Java). И всъщност основната им имплементация не е много сложна – имаме един речник, в който ключа е типът на зависимостта, а стойността е конкретният тип на разрешението й. Нека започнем така:

public class Container  
{  
*// дефинираме си речник за зависимостите ни*  
   private IDictionary<Type, Type> dependencies;  
  
   private ContainerOptions options;  
  
*// ContainerOptions е просто енумерация, която, въпреки че не използва много в текущия проект, бихме могли да я използваме за да конфигурираме допълнитено начина, по който разрешаваме зависимостите*  
   private const ContainerOptions defaultOptions = ContainerOptions.None;  
  
   public Container()  
    : this(defaultOptions)  
   {  
   }  
  
   public Container(ContainerOptions options)  
   {  
      this.options = options;  
      this.dependencies = [new](http://www.google.com/search?q=new+msdn.microsoft.com) Dictionary<Type, Type>();  
   }  
}

След като сме дефинирали основните ни променливи и конструктори, които ще използваме, ни трябва начин да регистрираме зависимости и как да те да бъдат разрешавани. Тоест, двойка типове, които ни показват кой клас да се инстанцира, когато ни е необходим конкретна имплементация на даден интерфейс.

public void RegisterType<TDependencyType, TResolveType>()  
            where TDependencyType : class  
            where TResolveType : class  
        {  
            *// регистрираме двойката зависимост, разрешение, като ги запишем в речника*  
            this.dependencies.Add([typeof](http://www.google.com/search?q=typeof+msdn.microsoft.com)(TDependencyType),[typeof](http://www.google.com/search?q=typeof+msdn.microsoft.com)(TResolveType));  
        }

Единственото, което ни остава е все пак да разрешим зависимостта. Ще го направим рекурсивно, използвайки алгоритъма за обхождане в дълбочина. По този начин ще си гарантираме, че ако някое от нашите разрешения има свои зависимости, бихме могли да ги разрешим и тях.

public T Resolve<T>() where T : class  
        {  
            var classType = [typeof](http://www.google.com/search?q=typeof+msdn.microsoft.com)(T);  
  
            var constructors = classType  
                .GetConstructors()  
                .OrderByDescending(x => x.GetParameters().Count());  
  
       *// класът, който се опитваме да инстанцираме трябва да има поне един конструктор*  
       *// ако няма, то за нас би било невъзможно да го инстанцираме (бихме могли например да проверим*  
       *// дали няма метод, който да връща инстанция, посредство Reflection)*  
            if (!constructors.Any())  
            {  
                throw [new](http://www.google.com/search?q=new+msdn.microsoft.com) ArgumentException("The class to be resolved does not have any public constructors!");  
            }  
  
       *// започваме да проверяме конструкторите на класът*  
            foreach (var constructor in constructors)  
            {  
                *// взимаме параметрите, които текущия конструктор изисква*  
                var parameters = constructor.GetParameters();  
  
                *// ако имаме работа с празен конструктор можем просто да инстанцираме обект от този клас*  
                if (parameters.Length == 0)  
                {  
                    var result = Activator.CreateInstance<T>();  
                    return result;  
                }  
                else  
                {  
                    *// създаваме си списък, където ще пазим инстанциите на нашите обекти*  
                    var parameterObjects = [new](http://www.google.com/search?q=new+msdn.microsoft.com) List<object>();  
  
                    *// ако конструктора все пак приема някакви параметри трябва да ги проверим един по един*  
                    foreach (var parameter in parameters)  
                    {  
                        var parameterType = parameter.ParameterType;  
  
         *// ако параметърът е примитив или ако има празен конструктор можем да го инстанцираме*  
         *// и да го запишем в списъка с параметри*  
         if (parameterType.IsPrimitive ||parameterType.GetConstructors().Any(x => !x.GetParameters().Any()))  
                        {  
                            var obj = Activator.CreateInstance(parameterType);  
                            parameterObjects.Add(obj);  
                        }  
         *// ако имаме работа с абстрактен параметър или интерфейс, ще трябва да го потърсим в*  
         *// нашия речник с регистрирани зависимости*  
                        else if (parameterType.IsAbstract ||parameterType.IsInterface)  
                        {  
                            var concreteObjectType =this.dependencies[parameterType];  
                            *// посредством Reflection извикваме рекурсивно методът ни за разрешаване на зависимости*  
                            *// на новата зависимост*  
                            var method =  
                                [typeof](http://www.google.com/search?q=typeof+msdn.microsoft.com)(Container)  
                                    .GetMethod("Resolve")  
                                    .MakeGenericMethod(concreteObjectType);  
  
                            var obj = method.Invoke(this, null);  
  
                            parameterObjects.Add(obj);  
                        }  
                    }  
  
          *// след като сме готови с всички параметри можем вече да извикаме конструктора*  
          *// и да инстанцираме обекта*  
                    var createdObject = (T)Activator.CreateInstance(classType, parameterObjects.ToArray());  
                    return createdObject;  
                }  
            }  
  
            throw [new](http://www.google.com/search?q=new+msdn.microsoft.com) Exception("Could not resolve the dependency");  
        }

Трябва, обаче да се има предвид, че това е непълна имплементация, която може да се разглежда като интересно упражнение и да изясни някои неясноти относно IoC контейнерите, по какъв начин действат и какво представлява dependency inversion. Пълният код може да бъде намерен на <https://github.com/csyntax/IocContainer>