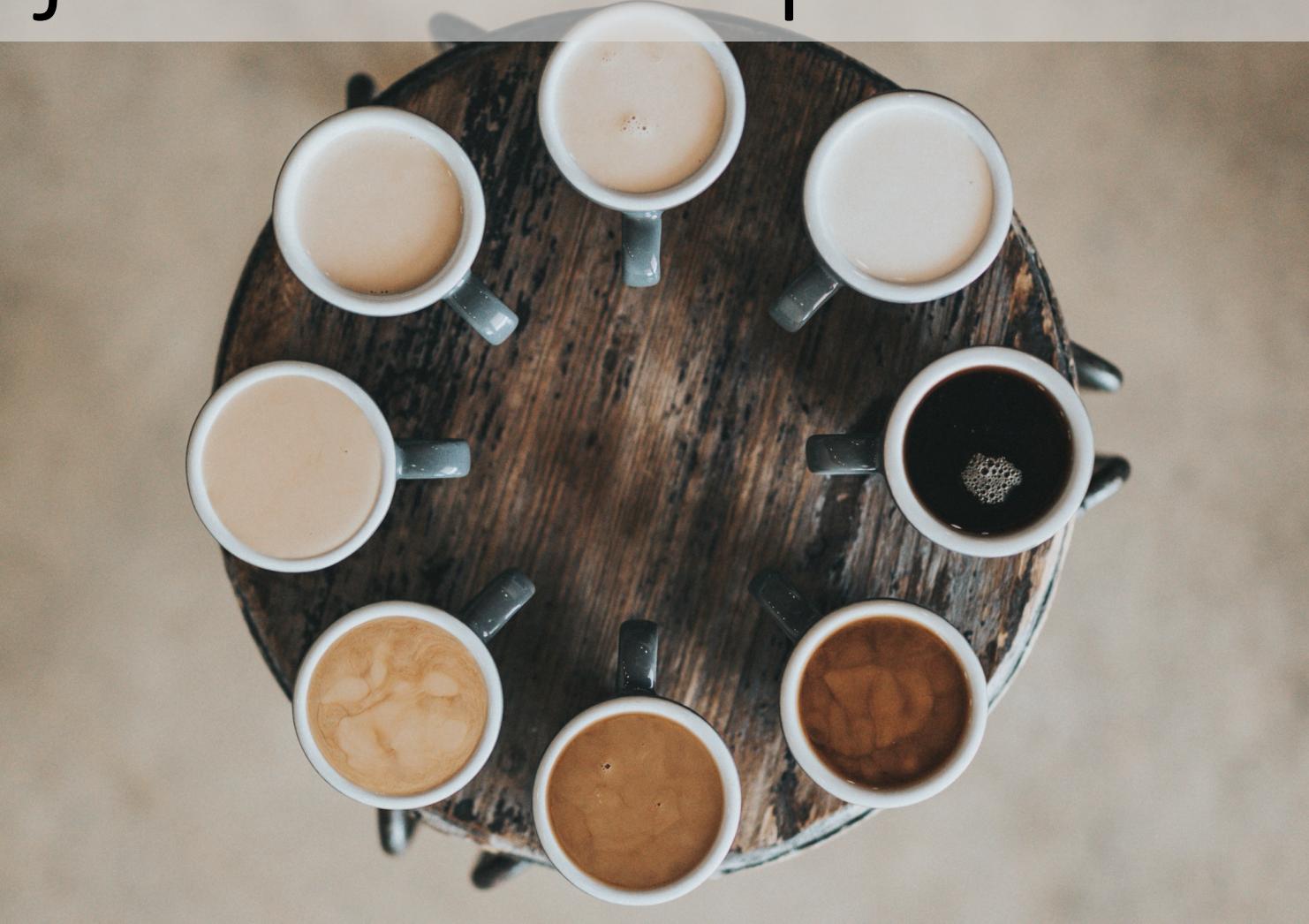


Injection de dépendances



Objectifs

- Définition de l'inversion de contrôle
- Cas particulier injection de dépendances
- Introduction au cadriciel (framework) d'injection de dépendances
« Unity container »

Inversion de contrôle – Définition

- « L'inversion de contrôle (inversion of control, IoC) est un patron d'architecture commun à tous les frameworks (ou cadre de développement et d'exécution). Il fonctionne selon le principe que le **flot d'exécution** d'un logiciel **n'est plus sous le contrôle** direct de **l'application** elle-même mais du framework ou de la couche logicielle sous-jacente. » (Wikipédia)
- WebForms, WinForms, ASP.Net MVC : le code est **appelé par le cadriel**
- De manière plus générale, une interface utilisateur programmée avec des événements : les développeurs ne contrôlent pas le déroulement de l'application, l'utilisateur oui

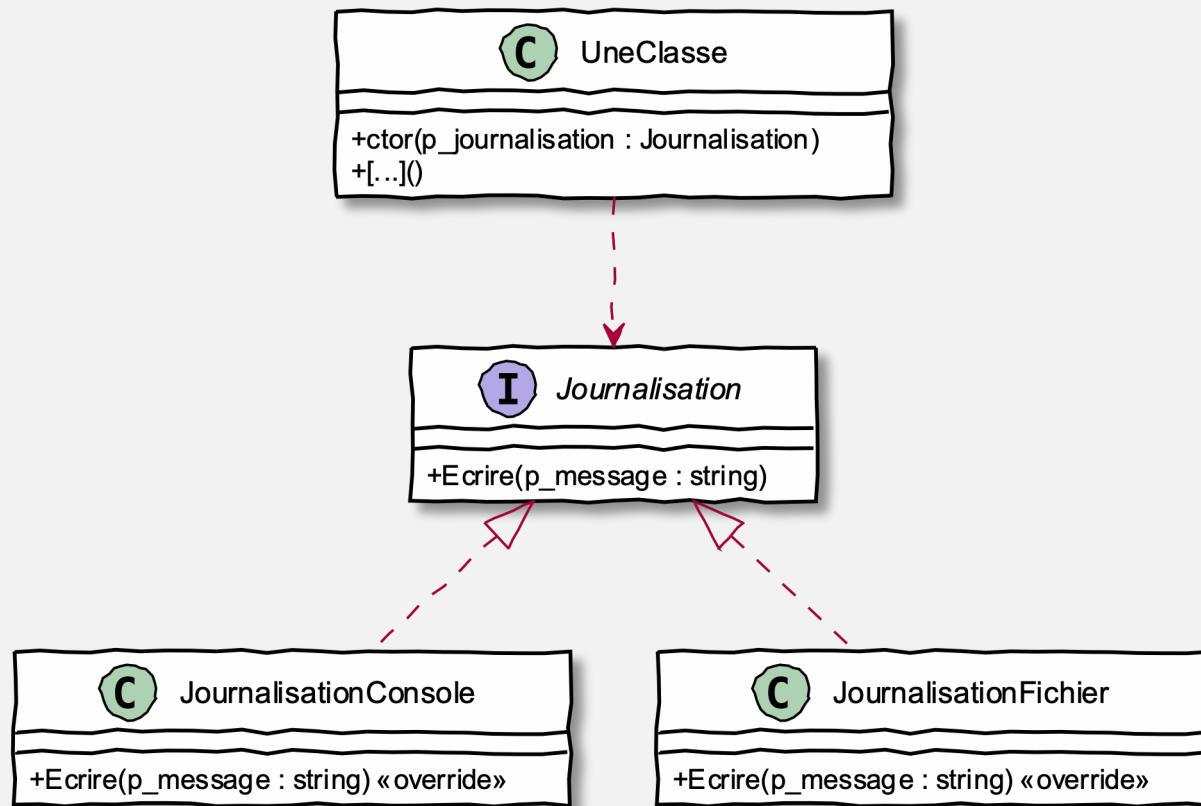
Injection de dépendances – Définition

- « **L'injection de dépendances** (dependency injection en anglais) est un mécanisme qui permet **d'implémenter** le principe de l'**inversion de contrôle**
Il consiste à **créer dynamiquement (injecter)** les **dépendances** entre les différents objets en **s'appuyant sur une description** (fichier de configuration ou métadonnées) ou de manière programmatique. Ainsi les **dépendances entre composants logiciels ne sont plus exprimées dans le code de manière statique mais déterminées dynamiquement à l'exécution.** » (Wikipédia)
- Dit plus vulgairement, avec l'injection de dépendances, **on ne veut peu d'utilisations de l'instruction « new »**

Injection de dépendances – Pourquoi

- Le but principal est de limiter les dépendances : Vous allez chercher à dépendre d'abstractions et non de classes concrètes
- Les applications vont être configurées pour associer des abstractions à des classes concrètes
- Exemple :
 - Vous pouvez avoir une interface « Journalisation »
 - Deux classes concrètes qui l'implantent :
 - Un qui utilise la sortie standard
 - Un qui utilise un fichier texte

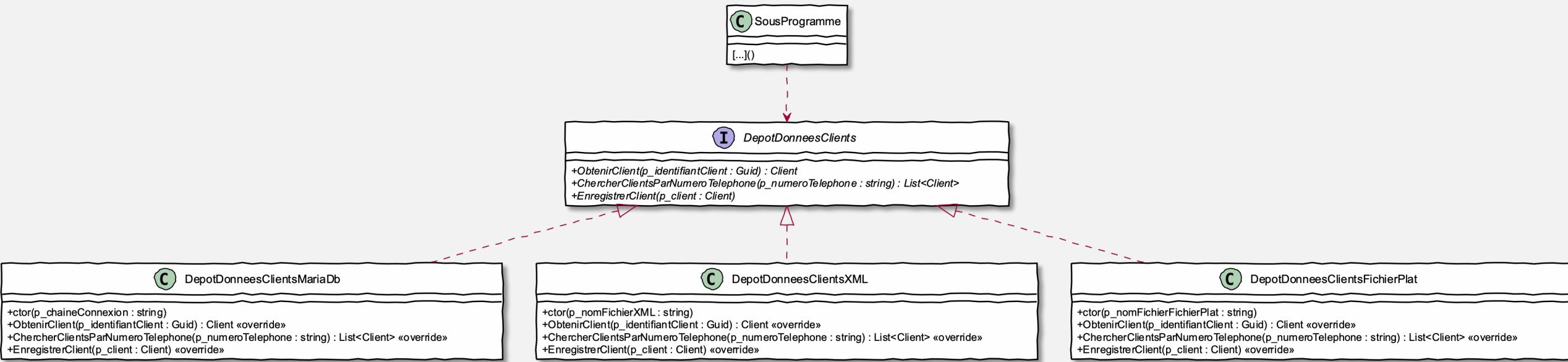
Injection de dépendances – Exemple journalisation



Injection de dépendances – Exemple dépôt de clients

- Vous pouvez avoir une interface « DepotDonneesClients »
- Trois classes concrètes qui l'implantent :
 - Un qui utilise le moteur de bases de données « MariaDb »
 - Un qui utilise le format de fichier plat (un client par ligne, champs définis par rapport à son numéro de colonnes)
 - Un qui utilise le format de données XML

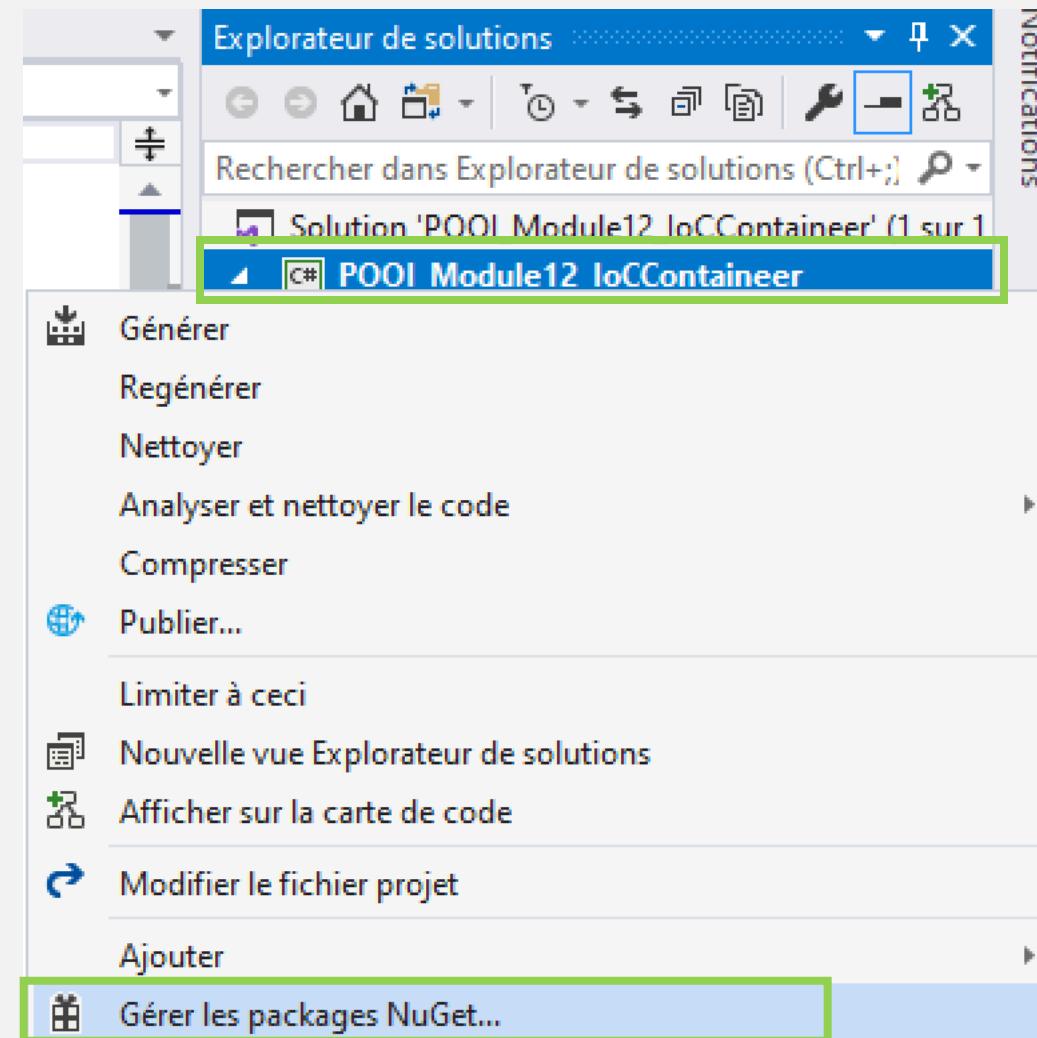
Injection de dépendances – Exemple dépôt de clients



Et C# dans tout cela ?

- En .Net, comme dans beaucoup de cadriel, il n'y a pas de bibliothèque imposée
- Beaucoup de cadriel intègrent leur bibliothèque d'injection de dépendances, soit en suggère une
- Dans notre cas, nous allons utiliser la bibliothèque « Unity container » de « Microsoft patterns and practices »
- « Unity container » peut injecter des dépendances dans :
 - Des constructeurs
 - Des propriétés
 - Des méthodes

Unity container – Installation dans le projet



Unity container – Installation dans le projet

NuGet : POOI_Module12_IoCContainerer ➔ X Program.cs

Parcourir Installé Mises à jour

Gestionnaire de package NuGet : POOI_Module12_IoCContainerer

Unity Inclure la version préliminaire

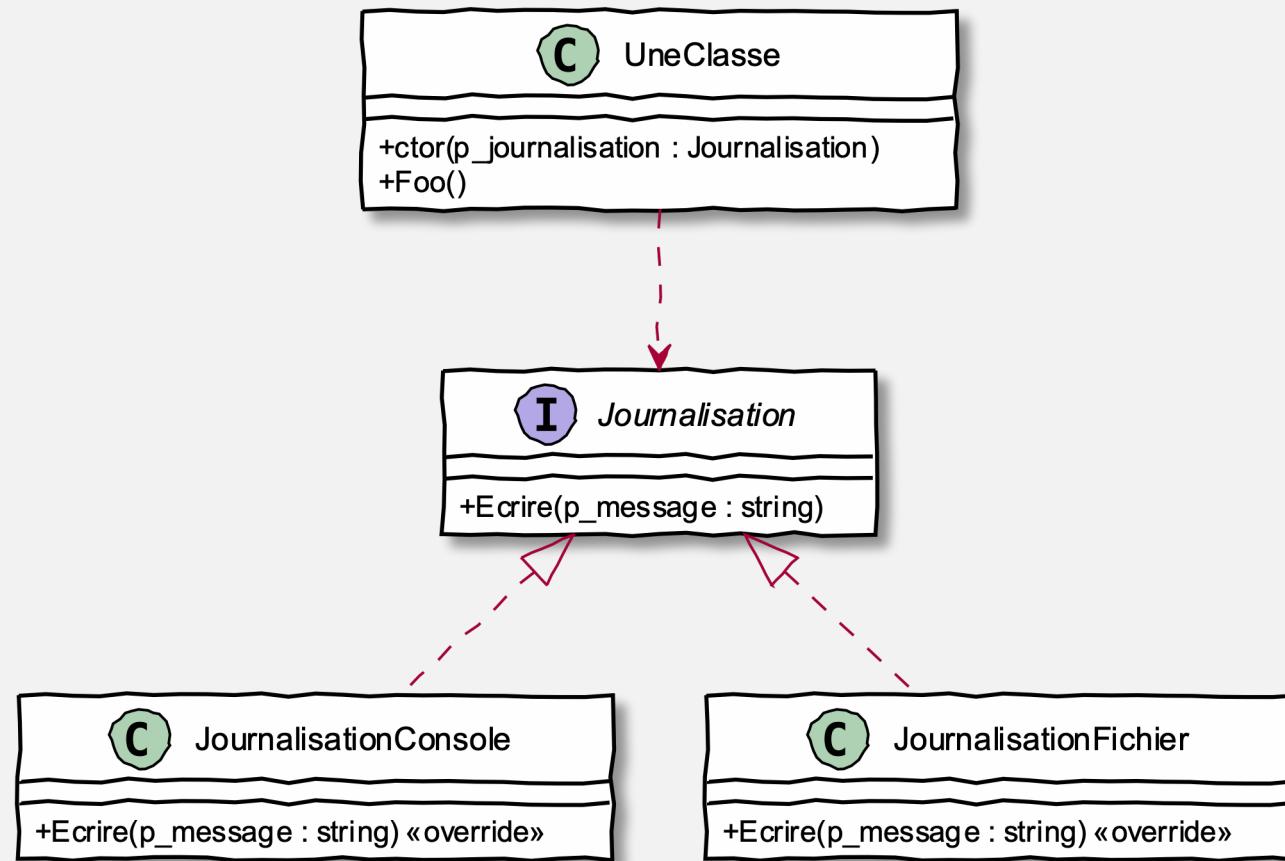
Source de package : nuget.org

 **Unity**  par Unity Container Project, 27,7M téléchargements
This package contains Unity Container and Abstractions libraries as a single package.

v5.11.6

 **Unity**   nuget.org
Version : Dernière version stable 5.11.6

Unity container – Exemple journalisation



Unity container – Préparation

```
public class UneClasse
{
    private IJournalisation m_journalisation;

    public UneClasse(IJournalisation p_journalisation)
    {
        this.m_journalisation = p_journalisation;
    }

    public void Foo()
    {
        // ...
        this.m_journalisation.Ecrire("Un message de Foo !");
    }
}
```

Unity container – Préparation

```
public interface IJournalisation
{
    void Ecrire(string p_message);
}
```

```
public class JournalisationConsole : IJournalisation
{
    public void Ecrire(string p_message)
    {
        Console.Out.WriteLine(p_message);
    }
}
```

Unity container – Préparation

```
public class JournalisationFichier : IJournalisation
{
    private static Object _lock = new Object();
    public void Ecrire(string p_message)
    {
        lock (_lock)
        {
            using (StreamWriter sw = new StreamWriter("journal.log"))
            {
                sw.WriteLine(p_message);
                sw.Flush();
            }
        }
    }
}
```

Utilisation classique

```
JournalisationConsole journalisationConsole = new Journalisation.JournalisationConsole();
journalisationConsole.Ecrire("Bonjour POOI !");
```

```
UneClasse uneClasse1 = new UneClasse(journalisationConsole);
uneClasse1.Foo();
```

```
JournalisationFichier journalisationFichier = new JournalisationFichier();
journalisationFichier.Ecrire("Bonjour POOI !");
```

```
UneClasse uneClasse2 = new UneClasse(journalisationFichier);
uneClasse2.Foo();
```

Unity container – Enregistrement et résolution

- « Unity container » doit connaître les classes concrètes associées aux abstractions. Les couples abstraction / classes concrètes doivent être enregistrée

```
IUnityContainer container = new UnityContainer();
container.RegisterType<IJournalisation, JournalisationConsole>();
```

Unity container – Enregistrement et résolution

- La résolution est l'action inverse : à partir du nom de l'abstraction, on obtient un objet

```
IJournalisation journalisation = container.Resolve<IJournalisation>();
```

```
UneClasse uneClasseIoC = container.Resolve<UneClasse>();  
uneClasseIoC.Foo();
```

« *UneClasse* » n'est pas enregistrée.

Ici, la méthode « *Resolve* » appelle le constructeur par initialisation et crée un objet correspondant à « *IJournalisation* ».

Unity container – Concrètement

- Enregistrement des couples abstractions / classes concrètes :
 - Créez une classe « *IoCUnityContainer* »
 - Déclarer la propriété statique « *Conteneur* »
 - Faites une méthode statique « *EnregistrerDependances* » qui enregistre les dépendances
- Au besoin, utilisez cette propriété statique dans votre code :

```
UneClasse uneClasse = IoCUnityContainer.Conteneur.Resolve<UneClasse>();
```

Unity container – Concrètement

```
public class IoCUnityContainer
{
    private static Object _lock = new Object();
    private static IUnityContainer _unityContainer = null;
    public static IUnityContainer Conteneur
    {
        get
        {
            if (_unityContainer == null)
            {
                lock (_lock)
                {
                    if (_unityContainer == null)
                    {
                        _unityContainer = new UnityContainer();
                        EnregisterDependances();
                    }
                }
            }

            return _unityContainer;
        }
    }
    private static void EnregisterDependances()
    {
        _unityContainer.RegisterType<IJournalisation, JournalisationConsole>();
        // ou _unityConteneur.RegisterType<IJournalisation, JournalisationFichier>();
    }
}
```

Unity container – Aller plus loin

- Documentation : [https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff660867\(v=pandp.20\)](https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff660867(v=pandp.20))
- Tutorial : <https://www.tutorialsteacher.com/ioc/unity-container>