

Programmation avancée en c#.NET

Ing.Meryem OUARRACHI

Plan du module

Programmation WEB

- ☐ Généralités outils Web
- ☐ ASP MVC
- ☐ ASP.Net core
- ☐ Angular

Génie logiciel en .Net

BI en Self Service

Programmation distribuée avancée

- ☐ Web API
- ☐ GraphQL

CHAPITRE 3:

Génie logiciel en .Net

Génie logiciel en .Net

-Le génie logiciel est une science qui s'intéresse aux procédures systématiques qui permettent d'arriver à créer des logiciels fiables qui correspondent aux attentes du client, avec un coût d'entretien réduit et de bonnes performances tout en respectant les délais et les coûts de construction.

Qualité logicielle=Qualité fonctionnelle+Qualité structurelle

Génie logiciel en .Net

Dans ce cours on va voir l'application des deux principes qui sont parmi les bonnes pratiques pour produire des logiciels fiables

IOC -Inversion Of Control-

Les tests unitaires

IOC

-Inversion Of Control-

Principe ouvert/fermé

-En POO, le principe ouvert/fermé affirme qu'une classe doit être à la fois ouverte (à l'extension) et fermée (à la modification):

- **Ouverte** : signifie qu'elle a la capacité d'être étendue.
- **Fermée**: signifie qu'elle ne peut être modifiée que par extension, sans modification de son code source.

Principe ouvert/fermé

Pourquoi ce principe?

1.La modification d'un code existant peut ne pas être possible.

2.La modification peut provoquer l'apparition de bugs dans cette partie du code. Il faut alors refaire la liste des tests de non régression, ce qui peut entraîner des surcoûts non négligeables.

Principe ouvert/fermé

Pourquoi ce principe?

3.la modification d'un code existant peut provoquer des problèmes bien plus graves qu'un bug en terme de maintenance: le code devient fragile, visqueux, rigide ou plus immobile :

Rigide	un petit changement induit une grande quantité de petit changements à d'autres endroits dans le logiciel
Fragile	un changement mineur provoque des bugs dans d'autres endroits du logiciel
Immobile	impossible de réutiliser une partie du logiciel pour effectuer d'autre tache dans le logiciel ou à l'extérieur de celui-ci.
visqueux	moins coûteux d'introduire du code de mauvaise qualité plutôt que du code respectant l'architecture d'origine.

Principe ouvert/fermé

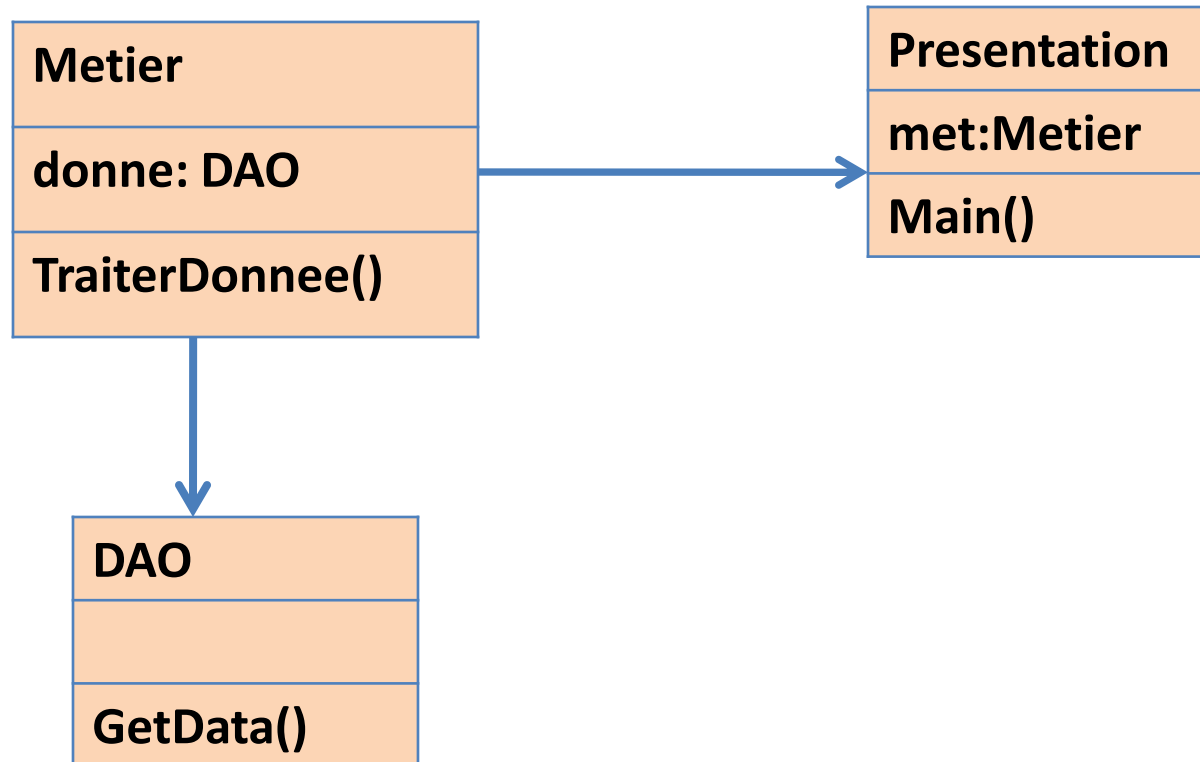
-L'idée est qu'une fois qu'une classe a été approuvée via des revues de code, des tests unitaires et d'autres procédures de qualification, elle ne doit plus être modifiée mais seulement étendue.

Le principe ouvert-fermé, est l'un des principe fondateur de l'architecture orientée objet. Son application a pour principe l'écriture d'un code source évolutif, dont les extensions ne touchent pas (ou peu) le code source déjà existant. On évite ainsi le pourrissement de l'architecture en garantissant que celle ci ne sera pas rigide/fragile/visqueuse ou immobile par l'intégration d'une nouvelle fonctionnalité.

Application de principe de ouvert/fermé

Exemple:

Soit l'application suivante qui implémente l'architecture 3-tiers suivante:



Application de principe de ouvert/fermé

Exemple:

```
class DAOImp
{
    public int GetDonne()
    {
        int x = 6;
        return x;
    }
}
```

```
class MetierImp
{
    DAOImp da;

    public MetierImp()
    {
        da = new DAOImp();
    }

    public int TraitementDonne()
    {
        int y = da.GetDonne() * 10;
        return y;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        MetierImp m = new MetierImp();
        Console.WriteLine("opération= " + m.TraitementDonne());
        Console.ReadKey();
    }
}
```

Problème du couplage fort

-Le couplage est une métrique indiquant le niveau d'interaction entre des classes ou des modules:


-Couplage fort: si les composants échangent beaucoup d'information.

-Couplage faible: si les composants échangent peu d'information.

Problème du couplage fort

-Dans l'exemple précédent, les classes DAOImpl/MetierImpl et les classes Presentation /MetierImpl sont liées par un couplage fort.

-Donc si on veut créer une nouvelle version de la méthode GetDonne() de la classe DAOImpl, nous serons obligés d'éditer le code source des classes DAOImpl et metierImpl

 On a pas respecté le principe « une application doit être fermée à la modification et ouverte à l'extension »



Rendre l'application faiblement couplé

Couplage faible

-Pour appliquer le couplage faible il faudra passer par les interfaces au lieu de passer par les classes directement.



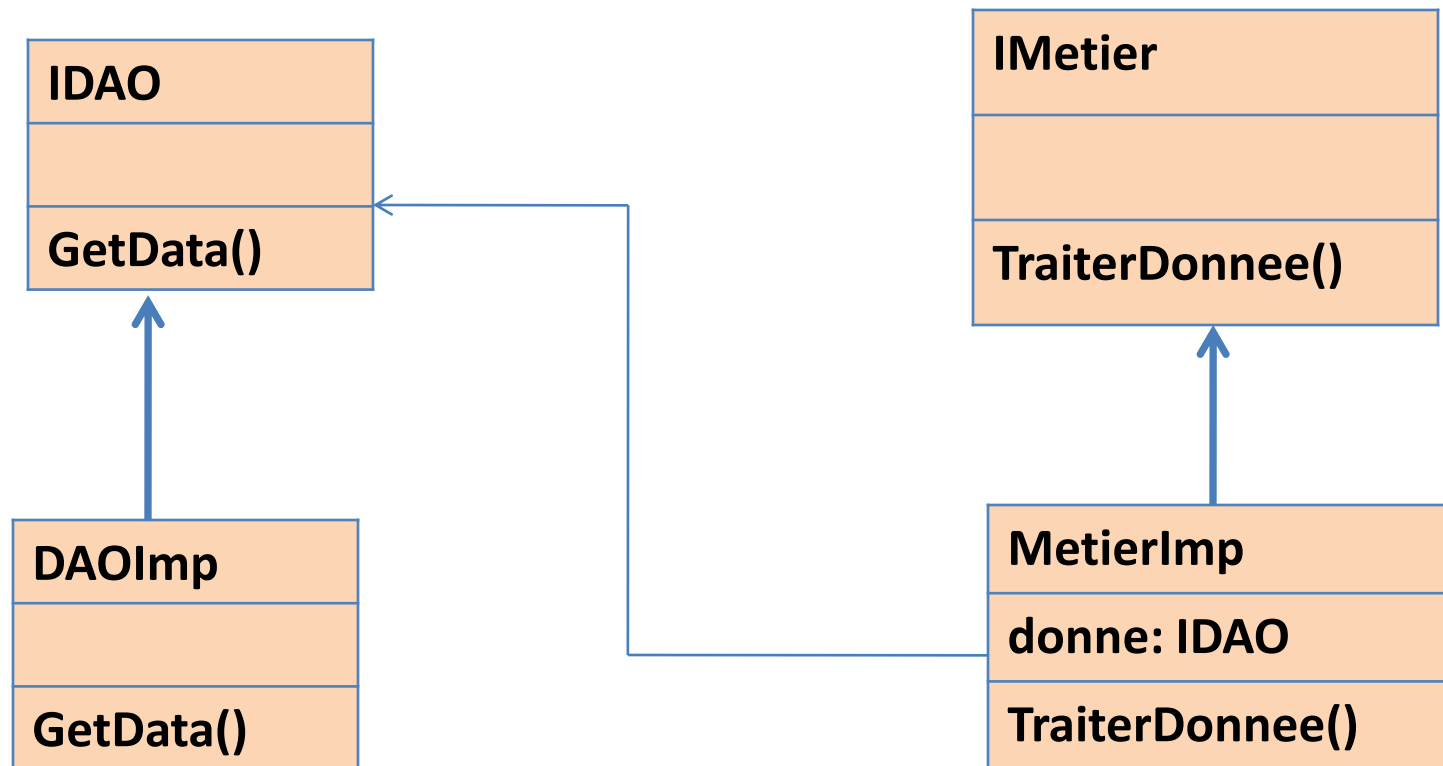
-la classe A peut fonctionner avec n'importe quelle classe qui implémente l'interface IB.

-Par la suite, n'importe quelle classe implémentant interface IB peut être associée à la classe A → le code de la classe A sera fermé à la modification

Couplage faible

-Pour appliquer le couplage faible il faudra passer par les interfaces au lieu de passer par les classes directement

-La nouvelle conception



Couplage faible

```
interface IDAO
{int GetDonne();
}
```

```
class DAOImpl:IDAO
{public int GetDonne()
{   int x = 6;
    return x;
}
}
```

```
interface IMetier
{   int TraitementDonne();
}
```

```
class MetierImp:IMetier
{   IDAO da { get; set; }

    public MetierImp()
    {   }
    public MetierImp(IDAO da)
    {   this.da = da;   }

    public int TraitementDonne()
    {
        int y = da.GetDonne() * 10;
        return y;
    }
}
```

Couplage faible

```
class Program
{
    static void Main(string[] args)
    {
        DAOImpl d = new DAOImpl();
        MetierImp m = new MetierImp(d);

        Console.WriteLine("opération= " + m.TraitementDonne());
        Console.ReadKey();
    }
}
```



On applique la notion de IOC

IOC -Inversion of control-

- L'inversion de contrôle est un patron d'architecture commun à tous les frameworks . 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.
- Il est connu aussi étant « *l'inversion de dépendances* » qui est un patron de conception permettant, en POO, de découpler les dépendances entre objets.

Inversion de dépendances

-L'inversion des dépendances est une implémentation de l'inversion de contrôle. Pour diminuer le couplage entre les classes, on va ajouter une interface entre chaque classe, de façon à ce qu'au lieu d'appeler une classe physique, l'appelant appelle une interface, ceci permettant d'ajouter un niveau d'abstraction supplémentaire entre l'appelant et l'appelé.

Inversion de dépendances

-Pour appliquer le principe de IOC ,on peut procéder de 3 façons:

- Instanciation statique
- Instanciation dynamique
- Utiliser des conteneurs IOC

Inversion de dépendances

- **Instanciation statique :**

- Appliquée dans l'exemple précédent

- **Ses limites:** l'application n'est pas fermée à la modification d'une manière définitive puisqu'on a accès au code source de la classe présentation afin de déterminer le type de la classe DAOImpl à utiliser par la suite

- Il faut trouver un moyen pour que le code source de la classe présentation reste aussi fermé à la modification



Utiliser l'Instanciation dynamique

Inversion de dépendances

- **Instanciación dinámica:**

-L'instanciación dinámica d'objet permet de créer un objet à un instant donné et de libérer cet objet lorsqu'il n'est plus utilisé.

-On l'utilise quand l'objet alloué n'est pas connu à l'avance ou qu'il n'est pas certains d'être utilisé, par contre on utilise statique dans tous les autres cas normaux.

Cas d'utilisation: nous voulons créer plusieurs implémentations de classes qui réalisent une même interface, et choisir à l'exécution quelle implémentation utiliser

Inversion de dépendances

- **Instanciation dynamique:**

- Instanciation statique: on utilise new

- Instanciation dynamique:

1. Obtient le type de l'objet à instancier

```
Type t = Type.GetType(String nom);
```

2. Crée une instance du type spécifié en utilisant le constructeur par défaut de ce type.

```
Activator.CreateInstance (Type)
```


Inversion de dépendances

- Appliquer l'instanciation dynamique:

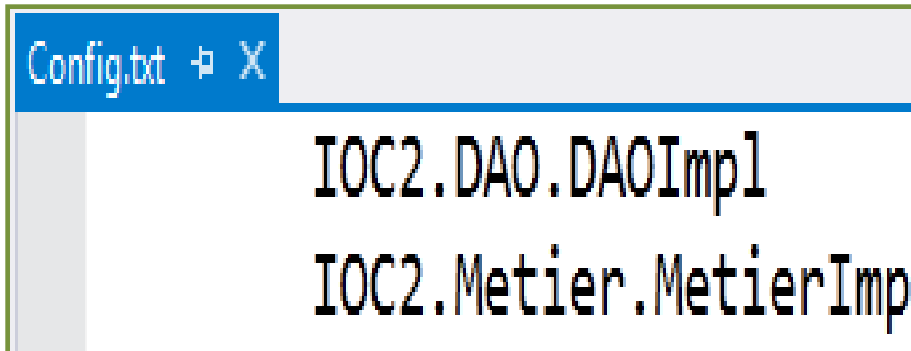
1. On crée un fichier de configuration contenant la liste des implémentations possibles

2. On fait des instanciations dynamiques qui se basent sur le fichier de configuration créé.

Inversion de dépendances

Exemple

1. Fichier de configuration



```
Config.txt  X
IOC2.DAO.DAOWImpl
IOC2.Metier.MetierImp
```

Inversion de dépendances

2. Instanciation dynamique:

```
static void Main(string[] args)
{
    //Lire le fichier
    string path = "Config.txt";
    string[] data = System.IO.File.ReadAllLines(path);
    string daoClassName = data[0];
    string metierClassName = data[1];

    //je vais connaitre le nom de la classe lors de l'exécution
    Type typeDao = Type.GetType(daoClassName);
    IDAO d = (IDAO)Activator.CreateInstance(typeDao);

    Type typeMet = Type.GetType(metierClassName);
    IMetier m = (IMetier)Activator.CreateInstance(typeMet, d);

    Console.WriteLine("opération d= " + m.TraitementDonne());
    Console.ReadKey();
}
```

Conteneurs IOC

- **Injection de dépendances**

l'injection de dépendances est une technique permettant de mettre en oeuvre l'inversion de dépendances.

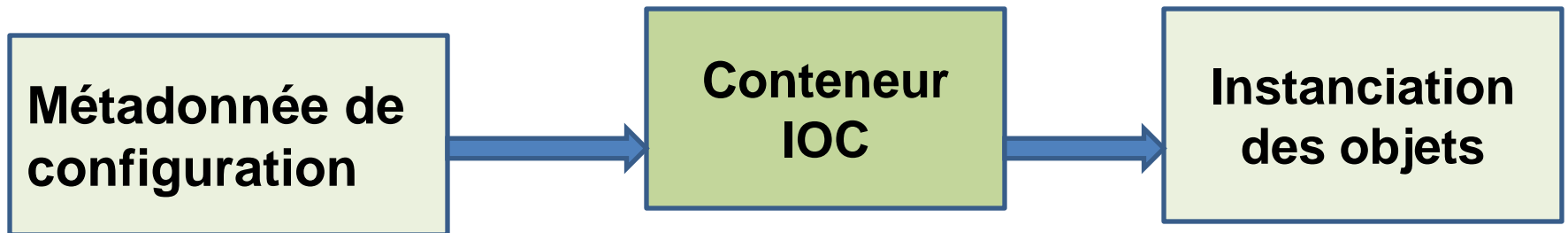
- **Conteneurs IOC:**

- Les conteneurs IoC sont des outils spécifiquement conçus pour faciliter l'injection de dépendances.

- Le conteneur récupère la responsabilité de l'instanciation des classes qu'elle doit injecter. On va voir comment ça passe dans le code.

Conteneurs IOC

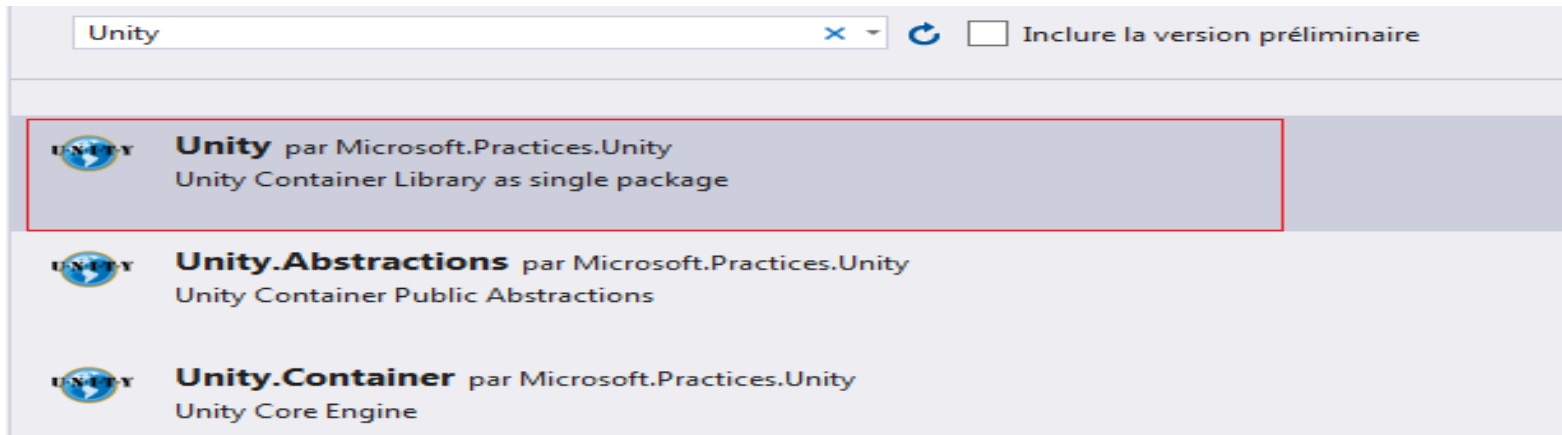
Le conteneur reçoit les instructions sur les objets à instancier, configurer, assembler par la lecture des métadonnées de configuration fournis. Les métadonnées de configuration peut être représentées soit par XML, annotations C#, ou un code C#.



Unity

-Unity est un conteneur IoC publié par Microsoft .Il facilite la construction d'applications faiblement couplées

•Installation:



•Après Installation

- Unity.Configuration
- Unity.Container
- Unity.Interception
- Unity.Interception.Configuration
- Unity.RegistrationByConvention

Unity

```
static void Main(string[] args)
{
    //Instancier le conteneur
    IUnityContainer unitycontainer = new UnityContainer();

    //Configuration de conteneur
    unitycontainer.RegisterType<IDAO, DAOImp>();

    //Résoudre la dépendance
    MetierImp appService = unitycontainer.Resolve<MetierImp>();

    //appel des méthodes
    Console.WriteLine("Valeur= " + appService.TraitementDonne());

    Console.ReadKey();
}
```

Unity

1. Instancier un objet de conteneur
 2. Configuration de dépendance: on ajoute la classe DAOImpl au conteneur
 3. Résoudre la dépendance en utilisant le conteneur IOC
 4. Maintenant on pourra appeler la fonction TraitementDonne qui appellera GetDonne de la classe DAOImp
- Remarque:** Il y'a 2 modes d'instanciation lors de configuration de dépendance

Unity

- **Mode d'instanciation:**

- **AddSingleton:**

Il crée une instance unique dans toute l'application. Il crée l'instance pour la première fois et réutilise le même objet dans tous les appels.

```
unitycontainer.RegisterSingleton<IDAO, DAOImp>();
```

Unity

- **Mode d'instanciation:**

- **AddTransient:**

Les objets auront une durée de vie transitoire, ce mode crée une nouvelle instance à chaque fois ces objets seront demandés

```
unitycontainer.RegisterType<IDAO, DAOImp>();
```

Unity MVC5

Exemple: On crée une application ASPMVC5

1.Ajout des services

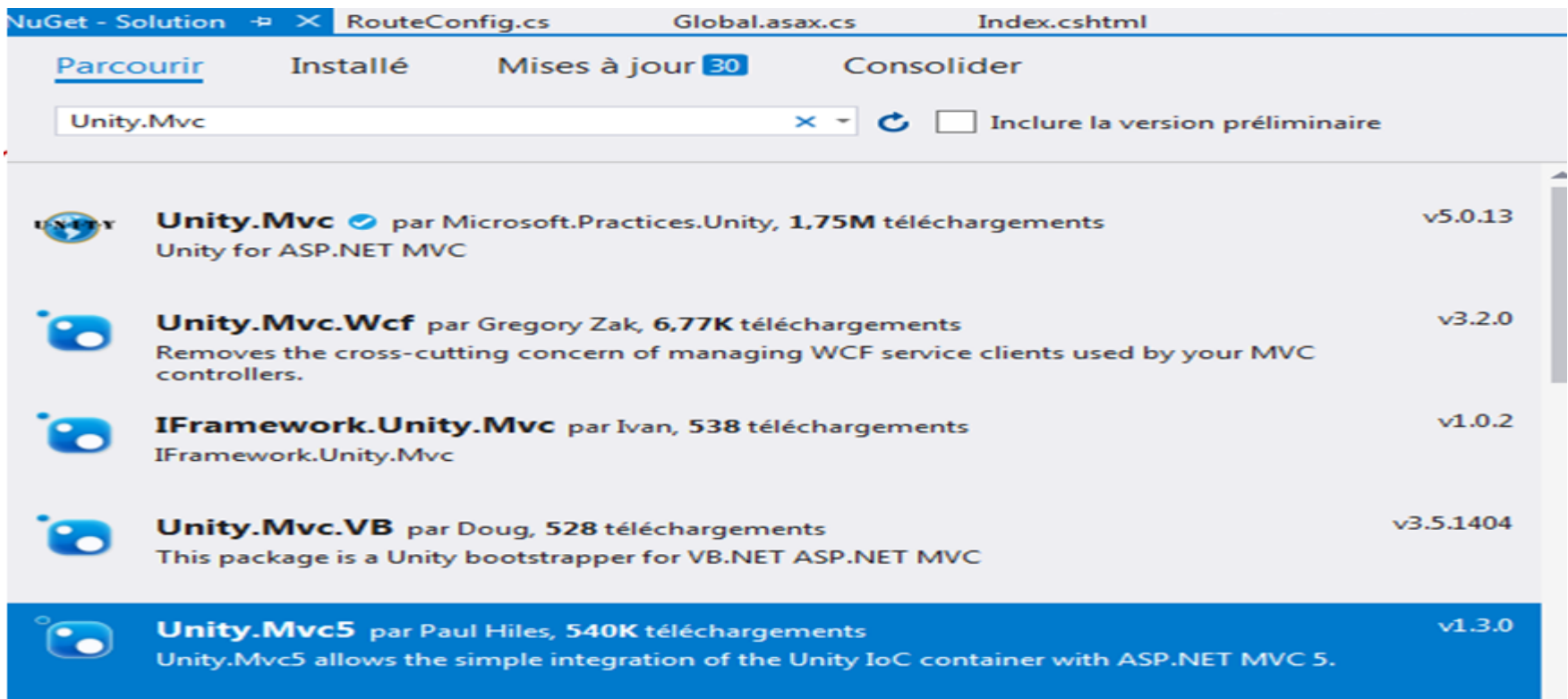
```
public interface IService1
{
    String GetValeurInstance();
}
```

```
public class Service1Imp:IService1
{
    public String GetValeurInstance()
    {
        return "Bonjour " + GetHashCode();
    }
}
```

Unity MVC5

Unity.Mvc5 est une bibliothèque qui permet l'intégration simple du conteneur Unity IoC de Microsoft avec ASP.NET MVC 5.

- **Installation:**



The screenshot shows the NuGet package manager interface with the search results for 'Unity.Mvc'. The interface includes tabs for 'Parcourir', 'Installé', 'Mises à jour 30', and 'Consolider'. A search bar contains 'Unity.Mvc' and a checkbox for 'Inclure la version préliminaire'. The results list several packages, with 'Unity.Mvc5' highlighted at the bottom.

Package Name	Author	Downloads	Version
Unity.Mvc	Microsoft.Practices.Unity	1,75M	v5.0.13
Unity.Mvc.Wcf	Gregory Zak	6,77K	v3.2.0
IFramework.Unity.Mvc	Ivan	538	v1.0.2
Unity.Mvc.VB	Doug	528	v3.5.1404
Unity.Mvc5	Paul Hiles	540K	v1.3.0

Unity MVC5

- **Configuration des services**

La configuration de l'injection de dépendance en ASP MVC se fait dans le fichier **UnityConfig.cs**

```
public static class UnityConfig
{
    public static void RegisterComponents()
    {
        var container = new UnityContainer();

        container.RegisterType<IService1, Service1Imp>();
        DependencyResolver.SetResolver(new UnityDependencyResolver(container));
    }
}
```

Unity MVC5

- **Configuration des services**

Ajoutez un appel à `UnityConfig.RegisterComponents()` dans la méthode `Application_Start` de **Global.asax.cs** pour que le framework MVC utilisera ensuite `Unity.Mvc5 DependencyResolver` pour résoudre les dépendances.

```
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);

    UnityConfig.RegisterComponents();
}
```

Unity MVC5

•Utilisation des services :Controller

```
public class Service1Controller : Controller
{
    private IService1 s1;
    private IService1 s2;

    public Service1Controller(IService1 s1, IService1 s2)
    {
        this.s1 = s1;
        this.s2 = s2;
    }

    public IActionResult Index()
    {
        ViewData["message"] = s1.GetValeurInstance() + " " + s2.GetValeurInstance();
        return View();
    }
}
```

Unity MVC5

- Utilisation des services :View

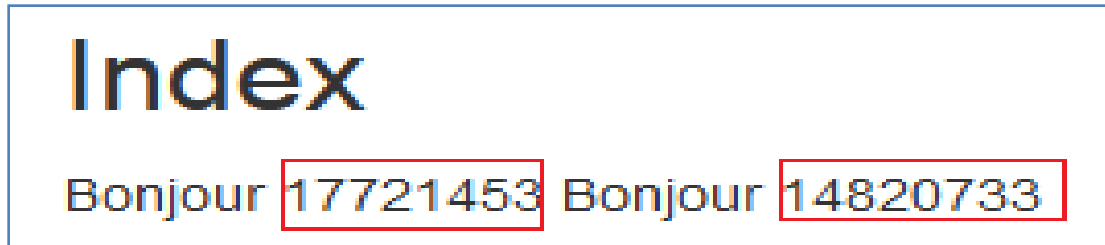
```
<h2>Index</h2>
```

```
@ViewData["message"]
```


Unity MVC5

- Résultat d'exécution:

-Mode Transient:

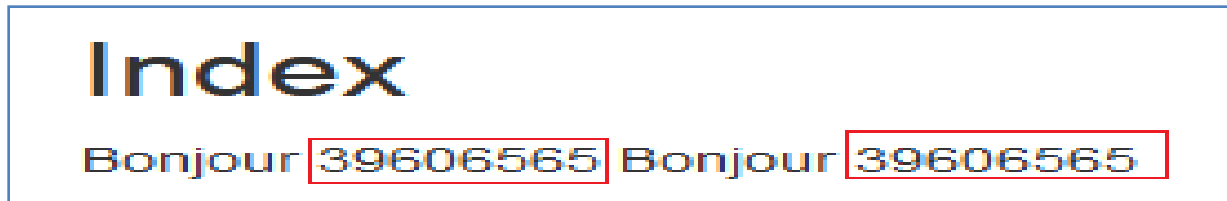


-On a une valeur pour chaque appel

Unity MVC5

- **Résultat d'exécution:**

- Mode singleton:**



- On a le même résultat pour les deux appels

- Aussi si on fait un refresh/ou on passe à un autre navigateur ,toujours on a le même résultat