



.NET - PRÉSENTATION

Arnaud.LEMETTRE@gmail.com

INTERVENANT



Arnaud LEMETTRE



MTI 2009

Société Alti :

- Expertise technique Microsoft

Professeur de .NET / WPF / Silverlight /
PLIC

Arnaud.LEMETTRE@gmail.com

PROGRAMME



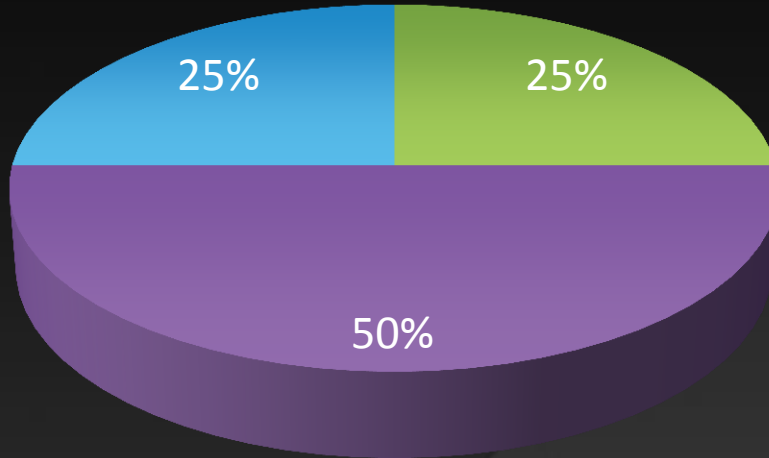
- ➡ Intégralité des cours sous forme de TP
- ➡ 16 TP jusqu'à mi juin
 - ▶ Maîtrise de l'environnement de développement (Visual Studio 2012)
 - ▶ SQL Server 2005 / 2008 (Edition Express) / 2012
 - ▶ Accès Base de données (ADO.net, Linq, EF)
 - ▶ Parsing XML
 - ▶ Open XML
 - ▶ ASP.Net
 - ▶ Webforms
 - ▶ WCF
 - ▶ WF
 - ▶ Analyse Applicative

PROGRAMME



➡ TP sous forme de mini projets
(gestion d'une bibliothèque, bug tracker, gestion de radar)

NOTATION



- TP
- Projet Libre
- QCM

NOTATION



- ➡ Les cours sont obligatoires :
 - ▶ Toute absence (non justifiée) équivaut à -0,5





DESCRIPTION

DESCRIPTION



➡ Framework est destiné à :

- ▶ Windows
- ▶ Windows Mobile
- ▶ Version légère avec moteur d'exécution pour navigateur Web (Silverlight)

➡ Le framework fournit :

- ▶ L'implémentation de la machine virtuelle compatible CLI
- ▶ Le framework .Net

DESCRIPTION



- ➞ CLI (Common Language Infrastructure)
 - ▶ Spécification ouverte développée par Microsoft
 - ▶ Description de l'environnement d'exécution

- ➞ Implémentation
 - ▶ Framework .Net
 - ▶ Silverlight / Moonlight
 - ▶ Mono (Novell)

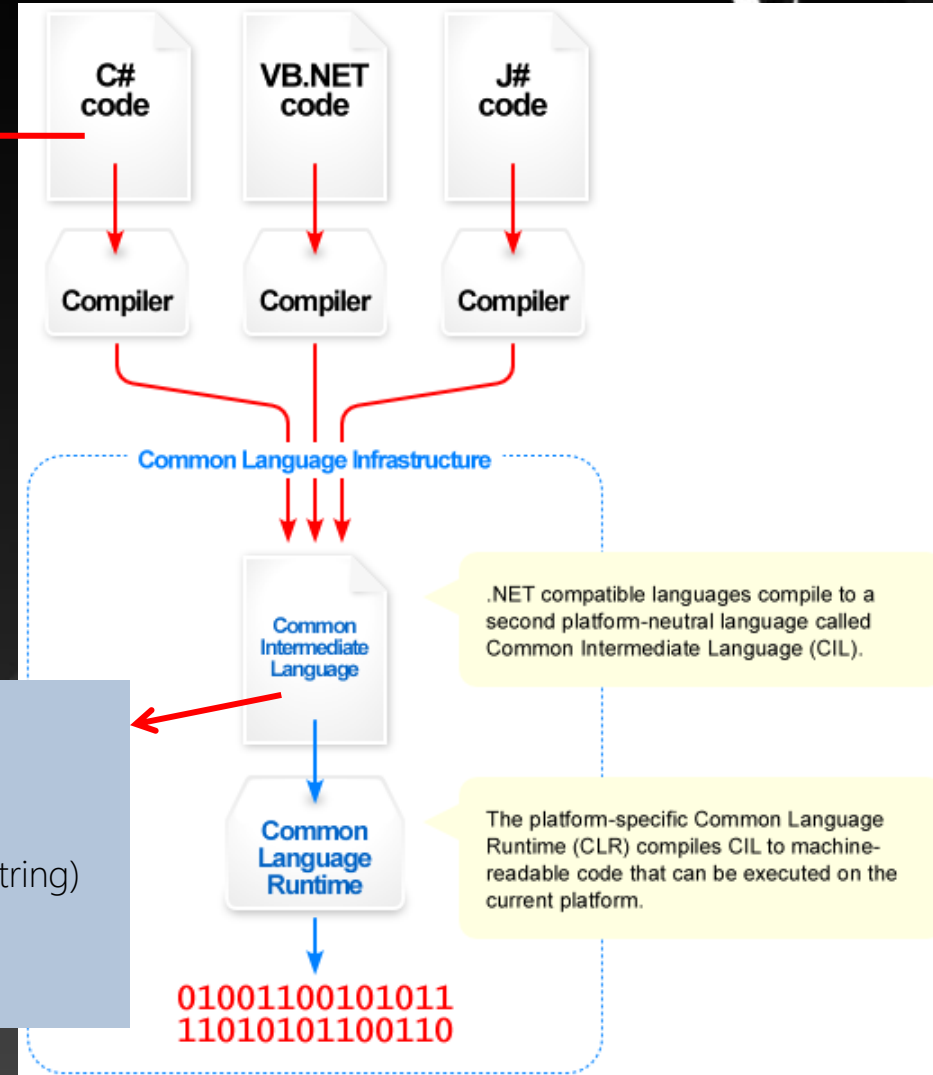
DESCRIPTION



```
using System;

class MainApp
{
    public static void Main()
    {
        Console.WriteLine("Hello World!");
    }
}
```

```
.method public hidebysig static void Main() cil managed
{
    /* 72 | (70)000001*/ ldstr "Hello World!"
    /* 28 | (0A)000002*/ call void System.Console::WriteLine(string)
    /* 2A |      */ ret
}
```



DESCRIPTION



- ➞ La machine virtuelle : CLR (Common Language Runtime)
 - ▶ Alloue la mémoire
 - ▶ Gère les droits des applications
 - ▶ Démarre et gère l' exécution
 - ▶ Ré allocation de la mémoire
- ➞ Interprétation du MSIL
- ➞ JIT (Just in Time Compiler)



HISTOIRE

HISTOIRE



- ➡ Fin 2000 : Sortie de la version Beta du Framework 1.0
- ➡ Début 2002 : Release du Framework 1.0
 - ▶ Version embryonnaire, peu développée, API légère
- ➡ Avril 2003 : Sortie du Framework 1.1
 - ▶ Support intégré pour les contrôles ASP.Net
 - ▶ Support ODBC et base de données Oracle
 - ▶ Ajout du framework .Net Compact pour Windows CE
 - ▶ Support de l'IPV6
 - ▶ Enrichissement de l'API

HISTOIRE

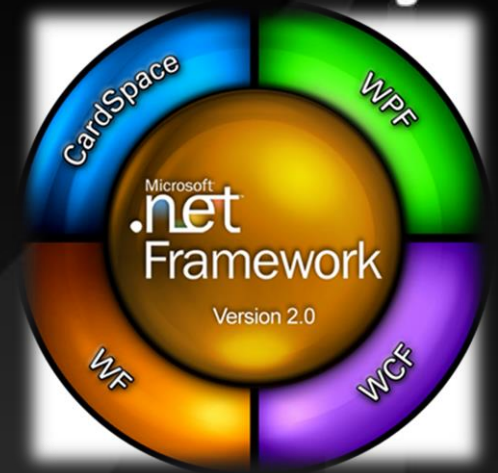


- ➡ Fin 2005 : Sortie du Framework 2.0
- ➡ Sortie avec :
 - ▶ Visual Studio 2005
 - ▶ SQL Server 2005
 - ▶ BizTalk Server 2006
- ➡ Ajout par rapport à la 1.1 :
 - ▶ Enrichissement de l'API
 - ▶ Support du 64 bits
 - ▶ Support des Generics
 - ▶ Nouveaux Web controls
 - ▶ Ajout des thèmes, skins, master page ...
 - ▶ Abstraction de la base de données

HISTOIRE



- ➡ Fin 2006 : Sortie du Framework 3.0
 - ▶ Peu de changement
 - ▶ Utilisation de la CLR du Framework 2.0
 - ▶ Pas de version compact



- ➡ WPF (Windows Presentation Foundation)
- ➡ WCF (Windows Communication Foundation)
- ➡ WF (Windows Workflow Foundation)
- ➡ CardSpace

HISTOIRE



- ➡ Novembre 2007 : Sortie du Framework 3.5
 - ▶ Toujours utilisation de la CLR 2.0
 - ▶ Inclusion du framework 2.0 SP1 (inclusion des nouvelles classes du framework)
 - ▶ Intégration de LinQ
 - ▶ Intégration du framework Ajax.Net

HISTOIRE



- ➡ 2010 sortie du Framework 4.0 et de visual studio 2010
- ➡ Ajout du DLR (Dynamic Language Runtime) coupe au dessus du CLR.
- ➡ Ajout de la programmation parallèle (Paralells Extensions)
- ➡ ...

HISTOIRE



- ➡ 2012 sortie du Framework 4.5 disponible et de visual studio 2012
 - ▶ Amélioration de la partie asynchrone (async / await, Dataflow)
 - ▶ ...



OFFRES MICROSOFT

VISUAL STUDIO



- ➡ IDE pour créer tout type de projet .Net
- ➡ Version actuelle : VS 2012
- ➡ Gère les langages :
 - ▶ C#
 - ▶ C++
 - ▶ Visual Basic
 - ▶ J#



SQL SERVER



- ➡ Système de gestion de base de données
- ➡ Version actuelle : SQL Server 2012
- ➡ Gestion des objets typiques
- ➡ Différents services



SHAREPOINT



- ➔ MOSS 2007/2010/2013 : Outil de portail et de travail collaboratif
 - ▶ Portail d'entreprise, portail web
 - ▶ Outils collaboratifs
 - ▶ Outils de portails
 - ▶ Moteur de recherche d'entreprise
 - ▶ Outils de gestion documentaire
 - ▶ Gestion de workflows
 - ▶ Gestion de publications

BIZTALK



- ➡ Microsoft BizTalk Server : serveur de gestion de processus métier
 - ▶ Adaptateurs pour communiquer entre différents logiciels
 - ▶ Automatisation processus métiers
 - ▶ Modélisation processus métiers
 - ▶ Plateforme d'asynchronisme

COMMERCE SERVER



➔ Microsoft Commerce Server

- ▶ Connexion application / système interne (connecteurs BizTalk)
- ▶ Gestion panier, historique de commande, adresse (livraison, facturation)
- ▶ Gestion d'un catalogue (stock, prix)
- ▶ Gestion des utilisateurs
- ▶ Rapports avancés d'e-commerce (taux de conversion, ventes ...)



WPF

WINDOW PRESENTATION FOUNDATION



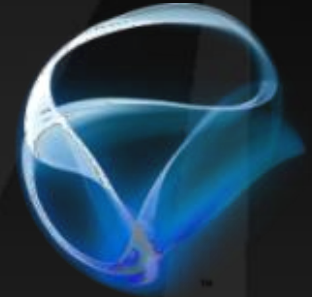
- ➞ Apparu dans .Net 3.0
- ➞ Destiné aux applications lourdes
- ➞ Meilleure utilisation des capacités de la carte graphique pour les applications lourdes (par exemple 3D)
- ➞ Développement : Visual Studio, Suite Expression
- ➞ Introduction du XAML



SILVERLIGHT



- ➡ Plugin pour navigateur web
- ➡ Applications riches dans un moteur de rendu vectoriel (basé sur WPF)
- ➡ Version actuelle : Silverlight 5.0
- ➡ Concurrents : Flash, Flex
- ➡ Développement : Visual Studio, Suite Expression



Microsoft®
Silverlight™

WCF

WINDOW COMMUNICATION FOUNDATION

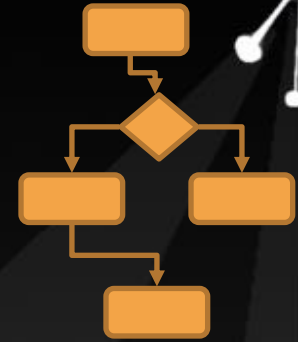


- ➡ Apparu dans .Net 3.0
- ➡ Permet d'unifier les communications entre différentes applications. (Com, Web service, Queue, P2P, ...)
- ➡ Abstraction du protocole d'échange : Séparation de la configuration réseau, du code.
- ➡ Facilité pour la sécurisation des échanges.

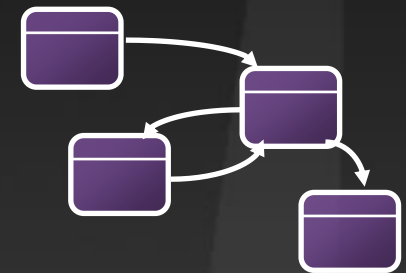


WF

WORKFLOW FOUNDATION



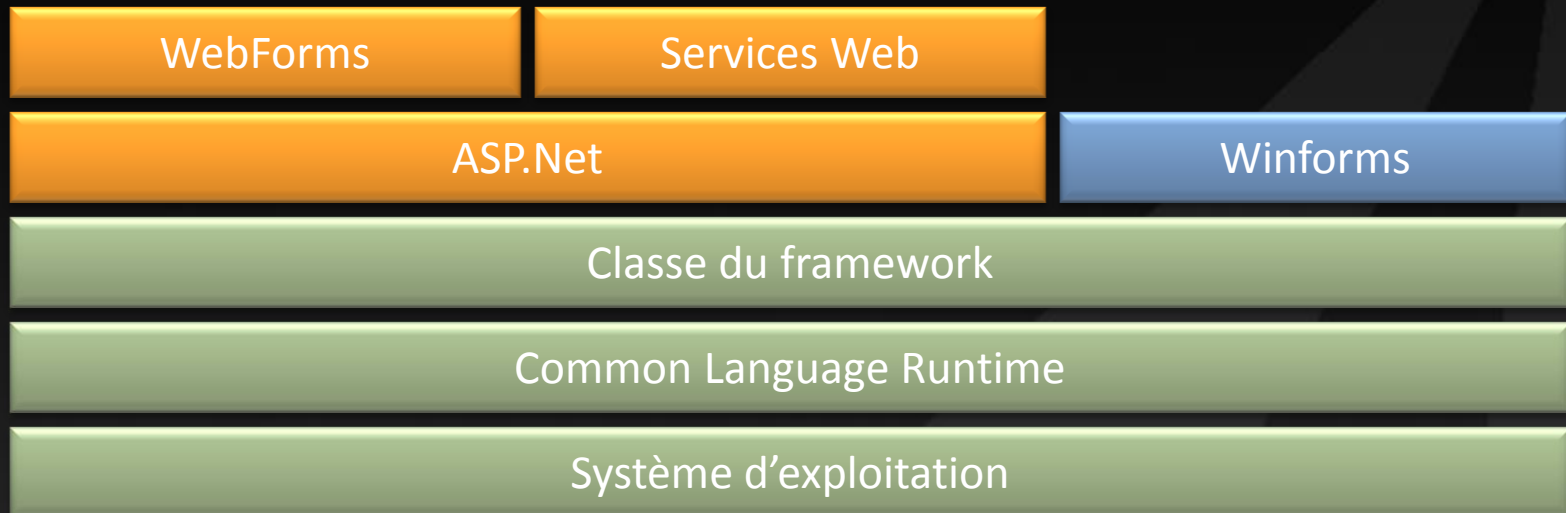
- ➞ Apparu dans .Net 3.0
- ➞ Moteur de Workflow (Flux de travail) propose 2 types : les machines d'états, et les workflows séquentiels.
- ➞ Permet donc la création et la modélisation de workflow.
- ➞ Il s'intègre à Visual Studio, et on dispose du designer





ASP.NET

ASP.NET

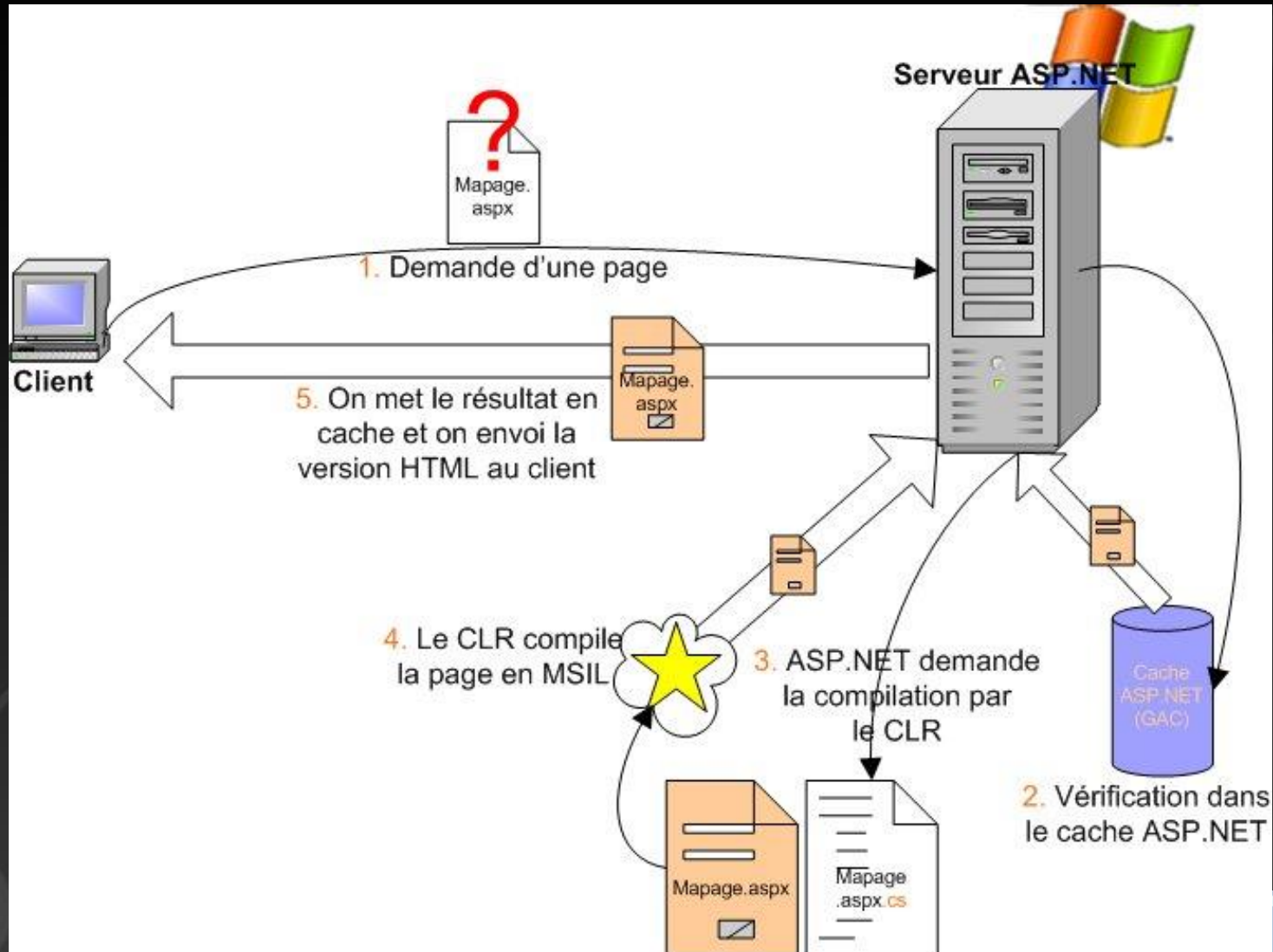


ASP.NET



- ➡ Langage événementiel
- ➡ Page ASPX / Code beside
- ➡ Cycle de vie d'une page
- ➡ Viewstate

ASP.NET



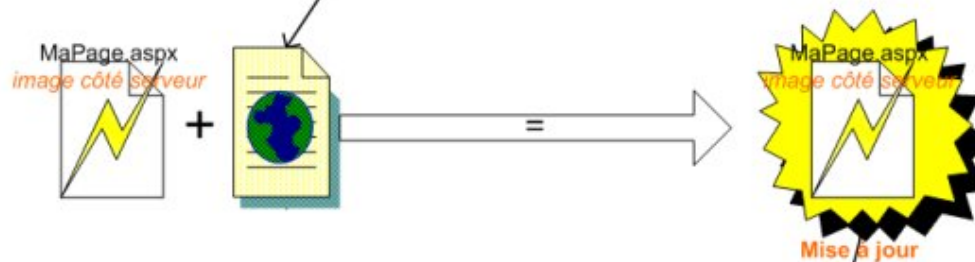
ASP.NET



1. Un *post-back* s'effectue :
le client envoie au serveur les champs de la page.



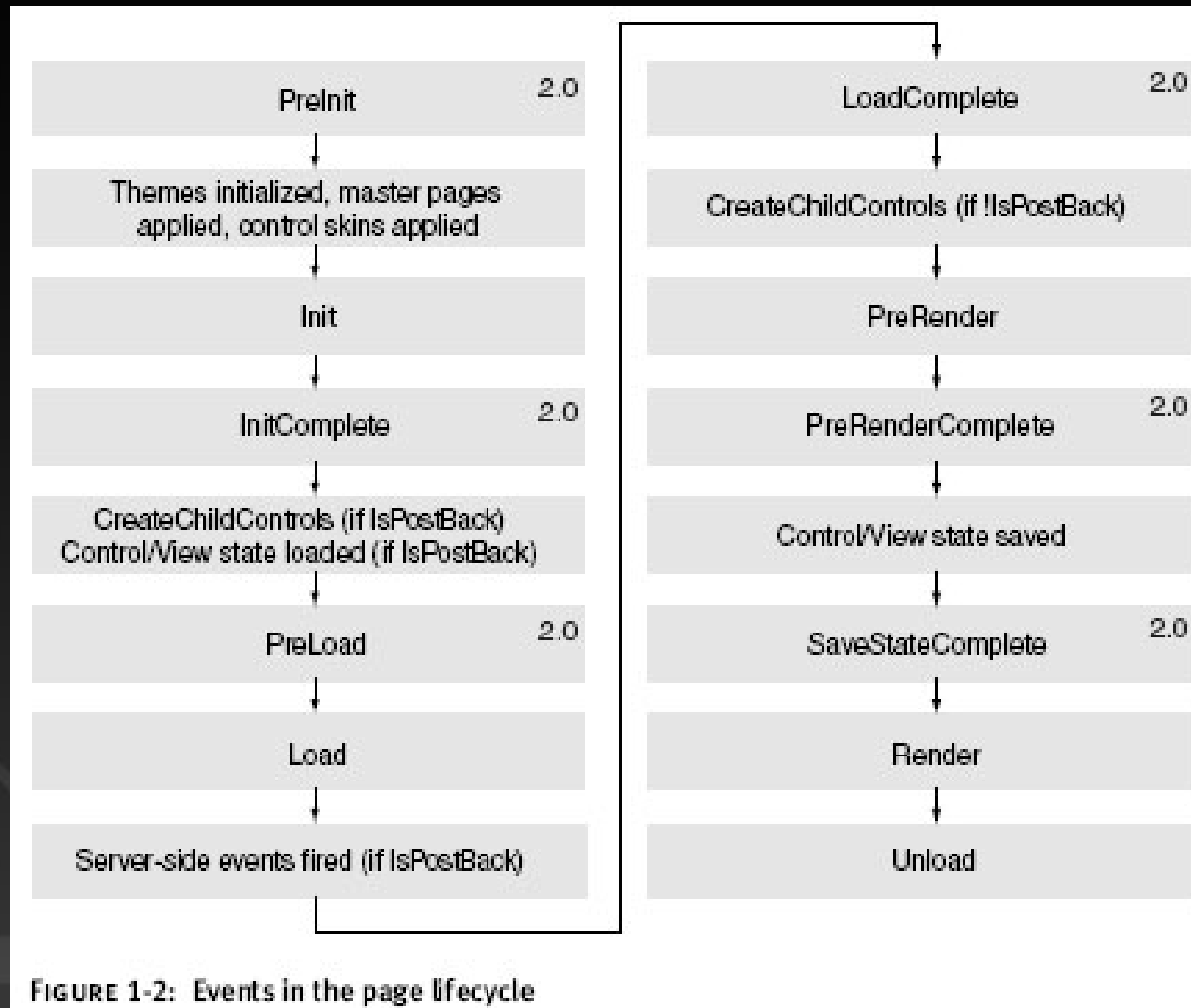
2. Côté serveur :
Avec l'image locale de la page et les champs envoyés par le client, on met à jour l'image locale.



3. On sérialise la page en HTML et on l'envoie au client



ASP.NET





LE LANGAGE C#

LE LANGAGE C#



- ➡ Représente la majorité des applications .Net
- ➡ Langage de programmation orienté objet
- ➡ Typage fort
- ➡ Très proche du langage C++
- ➡ Compilé en MSIL par le compilateur csc.exe (inclus dans le framework)
- ➡ Actuellement dans sa version 5.0

LE LANGAGE C#



Le .NET Framework utilise des espaces de noms pour organiser ses nombreuses classes.

Déclarer ses propres espaces de noms peut vous aider à contrôler la portée des noms de classes et de méthodes dans les projets de programmation plus volumineux.

Ils aident à la construction de programmes modulaires

Cela évite les « inévitables » conflits entre symboles homographes.

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;
```

```
namespace Sample.Demo1
```

```
{  
    class Program
```

```
{  
    // Ceci est un commentaire
```

```
    static void Main(string[] args)
```

```
    {  
        Console.WriteLine("Mon 1er programme .Net");  
        Console.WriteLine(DateTime.Now);
```

```
        Console.ReadLine();
```

```
    }  
}  
}
```

Namespaces utilisés

Namespace de l'application

Classe principale de l'application

Commentaire

Méthode principale

LE LANGAGE C#



➡ Les types simples :

- ▶ bool
- ▶ byte
- ▶ sbyte
- ▶ char
- ▶ decimal
- ▶ double
- ▶ float
- ▶ int
- ▶ uint
- ▶ long
- ▶ ulong
- ▶ object
- ▶ short
- ▶ ushort
- ▶ string

➡ Les types complexes :

- ▶ Array
- ▶ ArrayList
- ▶ List
- ▶ Hashtable
- ▶ Dictionary
- ▶ SortedList
- ▶ SortedDictionary
- ▶ Queue



Ces types peuvent être spécialisés, par exemple :
`List<string>`, `List<int>`, ...

LE LANGAGE C#



➡ Typage :

- ▶ Type de valeur (struct, enum)
- ▶ Type de référence (class, interface, delegate)

➡ Mot clé nullable

➡ Présentation des propriétés

- ▶ Mot clé « value »

LE LANGAGE C#



- ➡ Visibilité des méthodes / classes : public, private, protected, internal
- ➡ Héritage se définit par « : »
 - ▶ Mot clé : abstract, sealed, override
 - ▶ Héritage multiple interdit
 - ▶ Interfaçage multiple autorisé
 - ▶ Mot clé « base »

LE LANGAGE C#



➡ Propriétés diverses :

- ▶ Partial
- ▶ Abstract (classe et méthode)
- ▶ Extern (le code de la méthode n'est pas implémenté
[DllImport("avifil32.dll")]
private static extern void AVIFileInit();

LE LANGAGE C#



Exemple - soit les interfaces suivantes :

```
interface Interface1
{
    void method1();
}
```

```
interface Interface2
{
    void method2();
}
```

Exemple - soit la classe abstraite:

```
public abstract class AbstractClass
{
    public abstract bool AbstractMethod();
}
```

Implémentation de la
class1 héritant
de la classe abstraite
et implémentant les
deux interfaces

```
class Class1 : AbstractClass, Interface1, Interface2
{
    public override bool AbstractMethod()
    {
        throw new NotImplementedException();
    }

    public void method2()
    {
        throw new NotImplementedException();
    }

    public void method1()
    {
        throw new NotImplementedException();
    }
}
```

Multiple Interfaces

LE LANGAGE C#



➡ Présentation des boucles itératives

- ▶ For
- ▶ Foreach
- ▶ While
- ▶ Do ... while
- ▶ yield

➡ Bloc conditionnel

- ▶ If
- ▶ Switch ... case

Exemple d'utilisation du yield :

```
public class List
{
    public static IEnumerable Power(int number, int exponent)
    {
        int counter = 0;
        int result = 1;
        while (counter++ < exponent)
        {
            result = result * number;
            yield return result;
        }
    }

    static void Main()
    {
        // Display powers of 2 up to the exponent 8:
        foreach (int i in Power(2, 8))
        {
            Console.WriteLine("{0} ", i);
        }
    }
}
```

Sortie : 2 4 8 16 32 64 128 256

LE LANGAGE C#



- ➡ Les delegates
 - ▶ Pointeur de fonction
- ➡ Support du linq dans le langage
- ➡ Gestion des erreurs
 - ▶ Try ... catch
- ➡ Using (ne pas confondre avec le using pour les namespace)
 - ▶ <http://msdn.microsoft.com/en-us/library/yh598w02.aspx>

LE LANGAGE C#



➡ Les Exceptions

- ▶ Lorsqu'une exception se produit, le runtime informe l'utilisateur par le biais d'un message textuel de la nature de l'erreur et suggère une action pour résoudre le problème.
- ▶ La plupart des exceptions sont définies par le Framework, mais vous pouvez en créer.

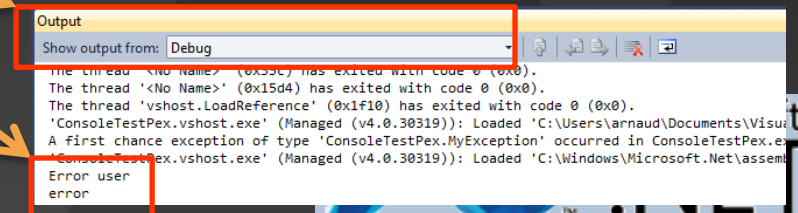
```
public class MyException : Exception
{
    public MyException(string message) : base(message)
    {
    }
}
```

```
static void Main(string[] args)
{
    try
    {
        throw new MyException("error");
    }
    catch (MyException ex)
    {
        Debug.WriteLine("Error user");
        Debug.WriteLine(ex.Message);
    }
    catch (Exception ex)
    {
        Debug.WriteLine("Error system");
        Debug.WriteLine(ex.Message);
    }
    finally
    {
        //toujours exécuté même si pas d'exception
    }
}
```

Déclenchement d'une exception

Toujours gérer l'exception la plus spécifique jusqu'à la plus généraliste

Affiche dans la sortie Debug :



LE LANGAGE C#



➡ Gestion du fileSystem

- ▶ 1 classe pour les dossiers : `System.IO.Directory`
 - ▶ `Exists`
 - ▶ `CreateDirectory`
 - ▶ `Delete`
 - ▶ `Move`
 - ▶ `GetFiles`

LE LANGAGE C#



➡ Gestion du fileSystem

- ▶ 1 classe pour les fichiers : `System.IO.File`
 - ▶ `Exists`
 - ▶ `OpenText`
 - ▶ `Delete`
 - ▶ `CreateText`
 - ▶ `Copy`

LE LANGAGE C#



➡ Gestion des fichiers

► Ecriture dans un fichier texte :

```
static void Main(string[] args)
{
    StreamWriter sw = new StreamWriter("fichier.txt");

    sw.WriteLine("première ligne ...");

    sw.Flush();
    sw.Close();
}
```

Pensez à bien fermer les flux, sinon toutes les données ne seront peut-être pas dans le fichier .

De plus si le fichier reste ouvert, vous ne pourrez pas l'utiliser ailleurs



Pensez à gérer les exceptions (fichier déjà utilisé, interdiction au niveau des droits ...)

LE LANGAGE C#



➡ Gestion des fichiers

► Lecture dans un fichier texte :

```
static void Main(string[] args)
{
    StreamReader sr = new StreamReader("fichier.txt");
    string ligne = sr.ReadLine();
    while (ligne != null)
    {
        Console.WriteLine(ligne);
        ligne = sr.ReadLine();
    }
    sr.Close();
}
```



Pensez à gérer les exceptions (fichier déjà utilisé, interdiction au niveau des droits ...)

LE LANGAGE C#



- ➡ Pour vous aider dans la navigation, il existe des classes :
- ▶ Environment.CurrentDirectory
 - ▶ Environment.SpecialFolder
 - ▶ Liste tous les dossiers particuliers de windows, ce qui évite les chemins codés en durs !
 - ▶ Path
 - ▶ Combine(...)
 - ▶ GetExtension
 - ▶ ...

LE LANGAGE C#



- ➡ En .Net, la mémoire est gérée par le garbage collector, ce qui nous évite de nous occuper de sa gestion. Un des désavantages du Garbage est qu'il s'exécute de façon « aléatoire » une fois qu'on n'utilise plus un objet, rien nous garantit qu'il sera enlevé de la mémoire tout de suite.
- ➡ Pour cela on peut aider le GC en utilisant IDisposable

LE LANGAGE C#



- ➔ Quand utiliser IDisposable ? :
- ▶ Lors de connexion sur une base de données
 - ▶ Utilisation de ressources (fichiers, ...)
 - ▶ Lors de communication avec Web Service
 - ▶ ...

LE LANGAGE C#



Implémentation d'IDisposable

```
class TemplateDisposable : IDisposable
{
    private StreamWriter _stream = null;
    ~TemplateDisposable()
    {
        Dispose(false);
    }
    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }
    protected virtual void Dispose(bool disposing)
    {
        if (disposing)
        {
            if (_stream != null)
            {
                _stream.Close();
                _stream.Dispose();
            }
        }
    }
}
```

Destructeur, pour les ressources natives (Facultatif)

Méthode appelée pour la libération des ressources

Libération des ressources managées

LE LANGAGE C#



Comment utiliser le dispose ?

```
TemplateDisposable dispo = new TemplateDisposable();  
dispo.Dispose();
```

En utilisant les using

```
using (TemplateDisposable dispo = new TemplateDisposable())  
{  
}
```

Comment fonctionne le using

```
TemplateDisposable dispo = new TemplateDisposable();  
try  
{  
    //do some work  
}  
finally  
{  
    if (dispo != null)  
        ((IDisposable)dispo).Dispose();  
}
```

LE LANGAGE C#



➞ Divers

- ▶ Using
- ▶ Namespace
- ▶ #region ... #endregion
- ▶ Attribut de méthode (ex : WebMethod)
- ▶ Surcharge d'opérateurs

LE LANGAGE C#



➞ Principe

- ▶ Les génériques permettent de paramétrer des classes, des structures, ..., et des méthodes par les types de données qu'ils stockent et manipulent
- ▶ Les génériques fournissent une vérification de type plus forte à la compilation
- ▶ D'un point de vue performance, les génériques évitent des opérations de vérification de type à l'exécution

```
private static void DisplayNotNull<E>(E value)
{
    if (value != null)
        Console.WriteLine(value);
    else
        Console.WriteLine("null value");
}
```

[http://msdn.microsoft.com/fr-fr/library/sz6zd40f\(v=vs.80\).aspx](http://msdn.microsoft.com/fr-fr/library/sz6zd40f(v=vs.80).aspx)

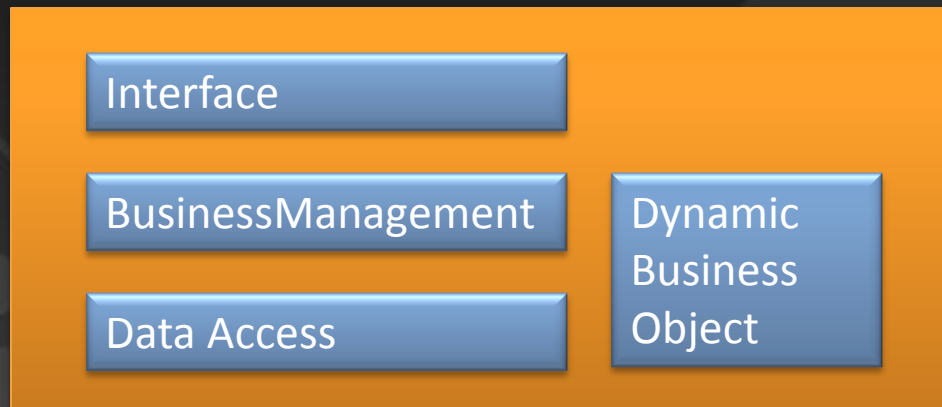
```
static void Main(string[] args)
{
    int? intValue = null;
    DisplayNotNull(12);
    DisplayNotNull(intValue);
}
```


LE LANGAGE C#



➡ Architecture n tier

- ▶ DAL : Data Access Layer
- ▶ BL : Business Layer
- ▶ UI : User Interface
- ▶ Entity layer (couche transversale)

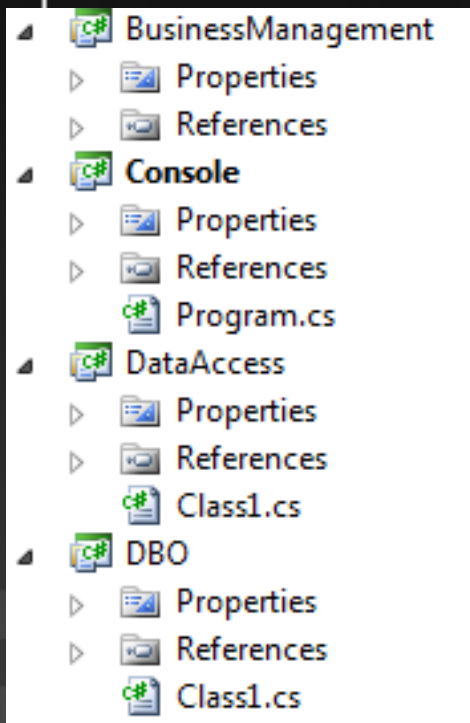


LE LANGAGE C#



➡ Et en concret ?

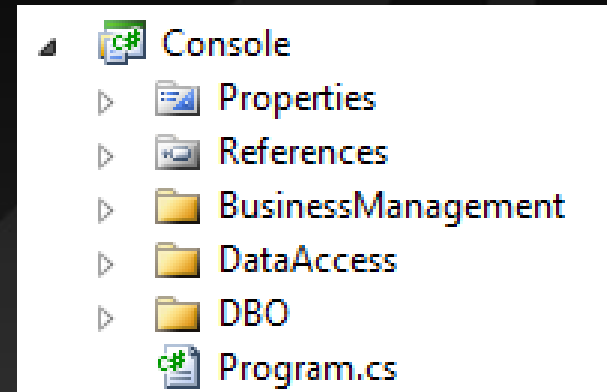
- ▶ Soit créer les couches sous forme de dossiers
- ▶ Soit sous forme de DLL



Comment choisir ?

Si vous devez réutiliser du code métier, alors il faut passer par une DLL. Par exemple : si vous devez utiliser des interfaces

différentes (Application console et application web) mais avec le même code

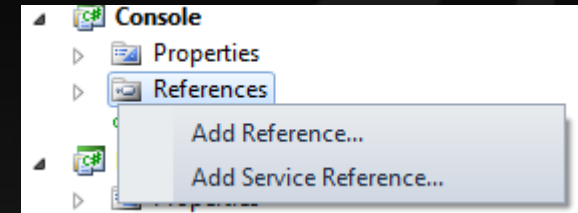
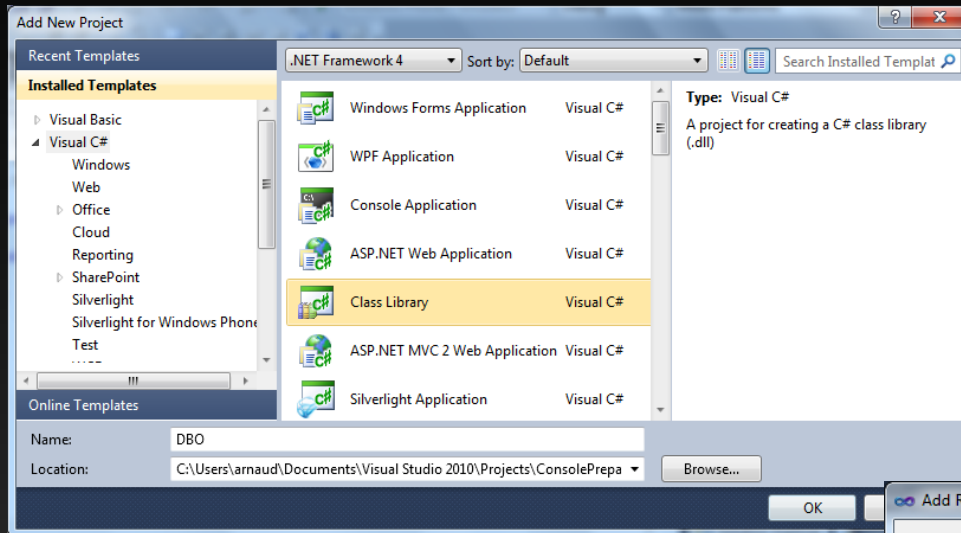


LE LANGAGE C#



➡ Comment créer une DLL ?

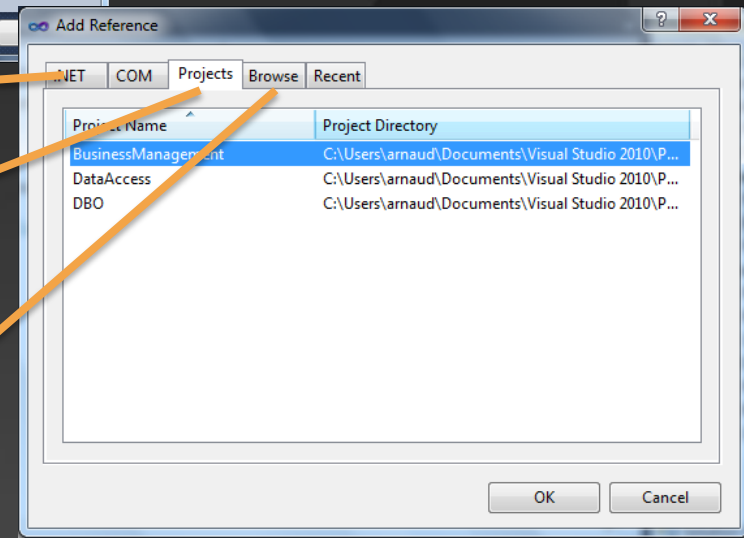
Clic droit sur une solution -> add -> new project



Assembly .NET

Projet dans la solution

DLL externe à la solution *.dll



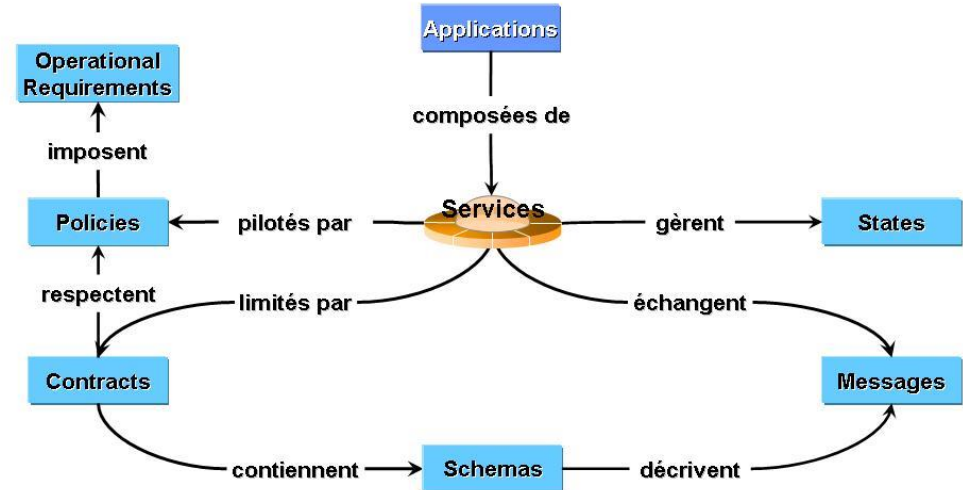
LE LANGAGE C#



➡ Architecture SOA

- ▶ La couche UI n'accède pas directement à la couche business

Les concepts de SOA

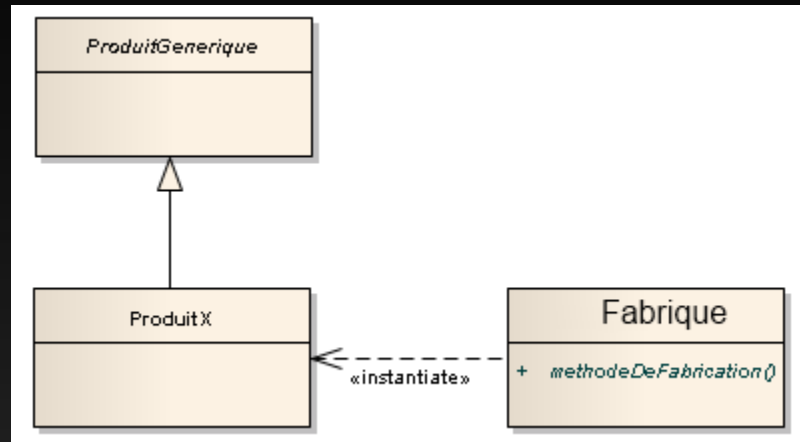


© Patrick Gantet, 2007

LE LANGAGE C#



➔ Design Pattern : *Factory*



Comme les autres modèles créationnels, la fabrique a pour rôle l'instanciation d'objets divers dont le type n'est pas prédéfini : les objets sont créés dynamiquement en fonction des paramètres passés à la fabrique. (Par exemple : des strings)

LE LANGAGE C#

➡ Design Pattern : Factory

Dans la réalité :

```
public class Fabrique
{
    public static IProduit GetAtelier(string typeName)
    {
        switch (typeName)
        {
            case "produitX":
                return new ProduitX();

            case "produitY":
                return new ProduitY();

            default:
                return new DefaultProduit();
        }
    }
}
```

Implémente
IProduit

Utilisation :

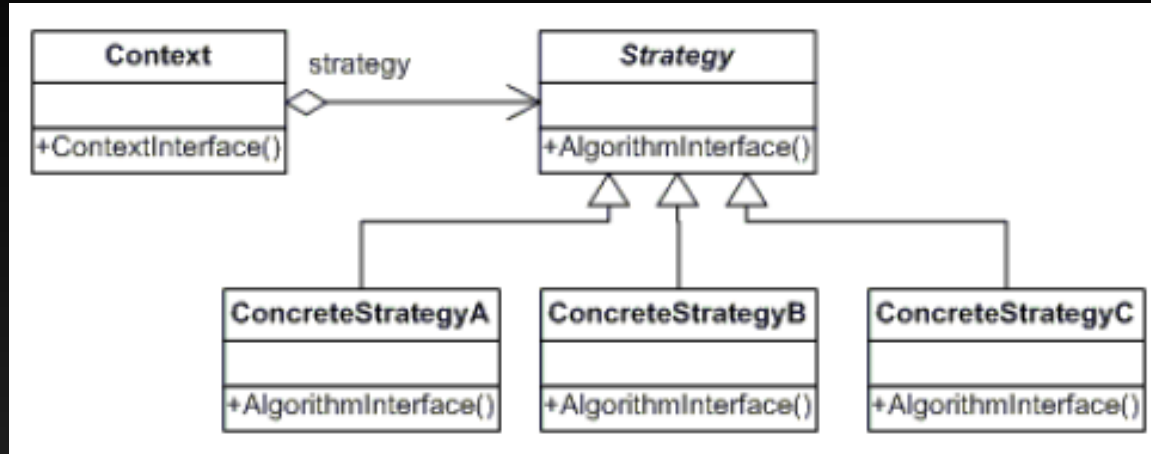
```
IProduit product = Fabrique.GetAtelier("ProduitX");
product.DoWork();
```



LE LANGAGE C#



➔ Design Pattern : *Strategy*



Solutionne le problème suivant :

Vous avez besoin que votre algorithme soit indépendant de votre classe qui l'utilise. Autrement dit il vous faut séparer votre algorithme (Strategy) de la classe.

Exemple : une troupe (des soldats) devant réagir sur un champ de bataille en appliquant soit une stratégie de défense ou d'attaque.

LE LANGAGE C#

Dans la réalité :



Context

Strategy

```
class Troupe
{
    ITroupeStrategy _strategy;
    public void SetStrategy(ITroupeStrategy strategy)
    {
        _strategy = strategy;
    }
    public void ExecuteStrategy()
    {
        _strategy.Move();
    }
}
```

```
interface ITroupeStrategy
{
    void Move();
}
```

Concrete Strategy

```
class AttackStrategy : ITroupeStrategy
{
    private int _x;
    private int _y;
    public void Move()
    {
        _x += 10;
        _y += 10;
    }
}
```

```
class DefendStrategy : ITroupeStrategy
{
    private int _x;
    private int _y;
    public void Move()
    {
        _x -= 10;
        _y -= 10;
    }
}
```


LE LANGUAGE C#



Utilisation :

```
static void Main(string[] args)
{
    bool attackEnemy = true;
    Troupe troupe = new Troupe();
    if (attackEnemy)
        troupe.SetStrategy(new DefendStrategy());
    else
        troupe.SetStrategy(new AttackStrategy());
}
```



MISE EN PRATIQUE

MISE EN PRATIQUE



- ➡ Installation / 1^{ère} utilisation de visual studio. [lien](#)
- ➡ Création d'un Hello World ☺
- ➡ Puis un mini jeu à faire sous console.
 - ▶ Le but de ce jeu sera de trouver le nombre choisi aléatoirement par l'ordinateur entre 0 et 100.
 - ▶ Quand l'utilisateur a trouvé, le programme propose de rejouer ou d'arrêter.