# .Net – Présentation

Arnaud.LEMETTRE@gmail.com

ÉCOLE D'INGÉNIEURS EN INFORMATIQUE

Microsoft® .NET

# Intervenant

- Arnaud LEMETTRE
  - MTI Promo 2009
  - Société Alti
    - Cellule R&D
    - Expert technique
  - Finaliste Imagine Cup 2008
  - Prof à l'EPITA

# Programme

- Intégralité des cours sous forme de TP
- 16 TP jusqu'à mi juin
  - Maitrise de l'environnement de développement (Visual Studio 2010)
  - SQL Server 2005 / 2008 (Edition Express)
  - Accès Base de données (ADO.net, LinQ, EF)
  - Parsing XML
  - Open XML

# Programme

- ASP.Net
- Webforms
- WCF
- WF
- Analyse Applicative

- TP sous forme de mini projets

(gestion d'une bibliothèque, bug tracker, gestion de radar)

# NOTATION

- Notation via TP (1/4 de la note de projet)
- Projet libre (3/4 de la note de projet)
- QCM (notation à part)

# DESCRIPTION


# Description

- Framework 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

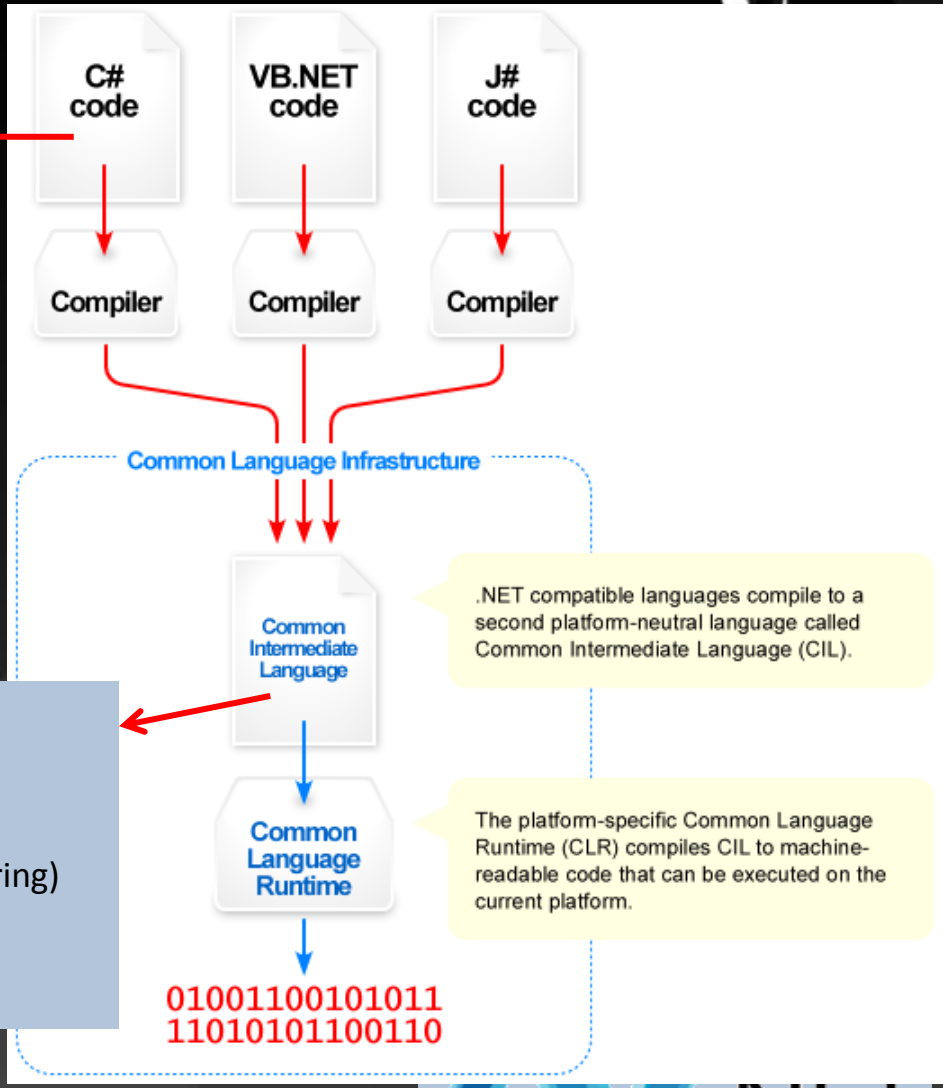
# Description

- 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

# Histoire

- 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

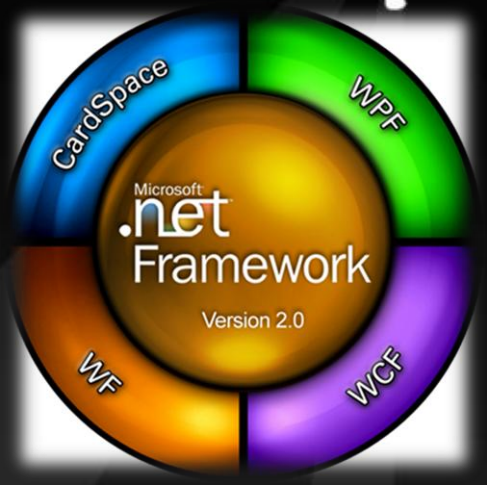
# Histoire

- Fin 2005 : Sortie du Framework 2.0
- Sortie avec :
  - Visual Studio 2005
  - SQL Server 2005
  - BizTalk Server 2006

# Histoire

- 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

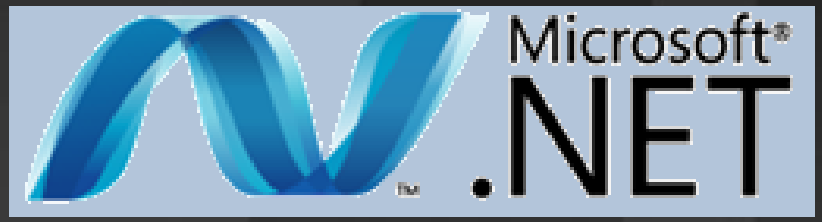
- WPF (Windows Presentation Foundation)
- WCF (Windows Communication Foundation)
- WF (Windows Workflow Foundation)
- CardSpace

# 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 disponible et de visual studio 2010
- Ajout du DLR (Dynamic Langage Runtime) coupe au dessus du CLR.
- Ajout de la programmation parallèle (Paralells Extensions)
- …




# OFFRES MICROSOFT


# Visual Studio

- IDE pour créer tout type de projet .Net
- Version actuelle : VS 2010
- Gère les langages :
  - C#
  - C++
  - Visual Basic
  - J#

# SQL Server

- Système de gestion de base de données
- Version actuelle : SQL Server 2008
- Gestion des objets typiques
- Différents services

# SHAREPOINT

- MOSS 2007/2010 : 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 workflow
  - Gestion de publication

# 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

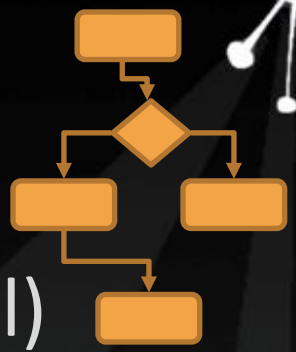
# Silverlight

- Plugin pour navigateur web
- Applications riches dans un moteur de rendu vectoriel (basé sur WPF)
- Version actuelle : Silverlight 4.0
- Version en RC : Silverlight 5.0
- Concurrents : Flash, Flex
- Développement : Visual Studio, Suite Expression

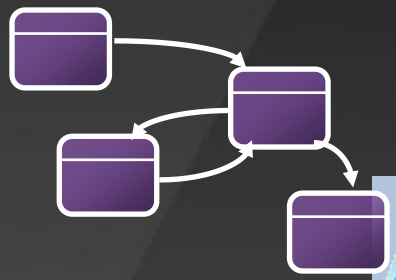
# WCF
# Window Communication Foundation

- 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 modélisation de workflow.
- Il s'intègre à Visual Studio, et on dispose du designer



# ASP.NET


# ASP.NET

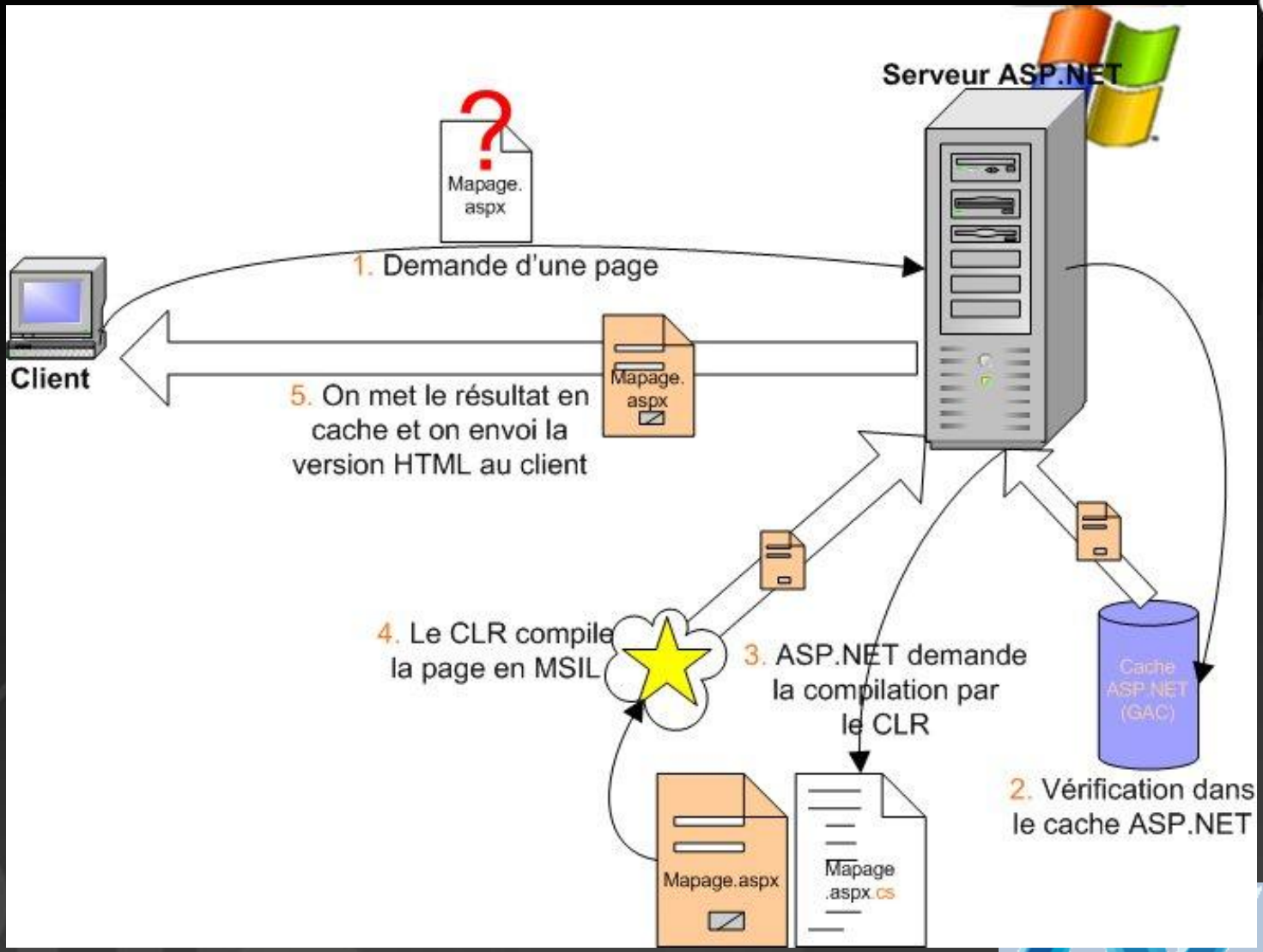

WebForms
Services Web
ASP.Net
Winforms
Classe du framework
Common Language Runtime
Système d'exploitation

# ASP.NET

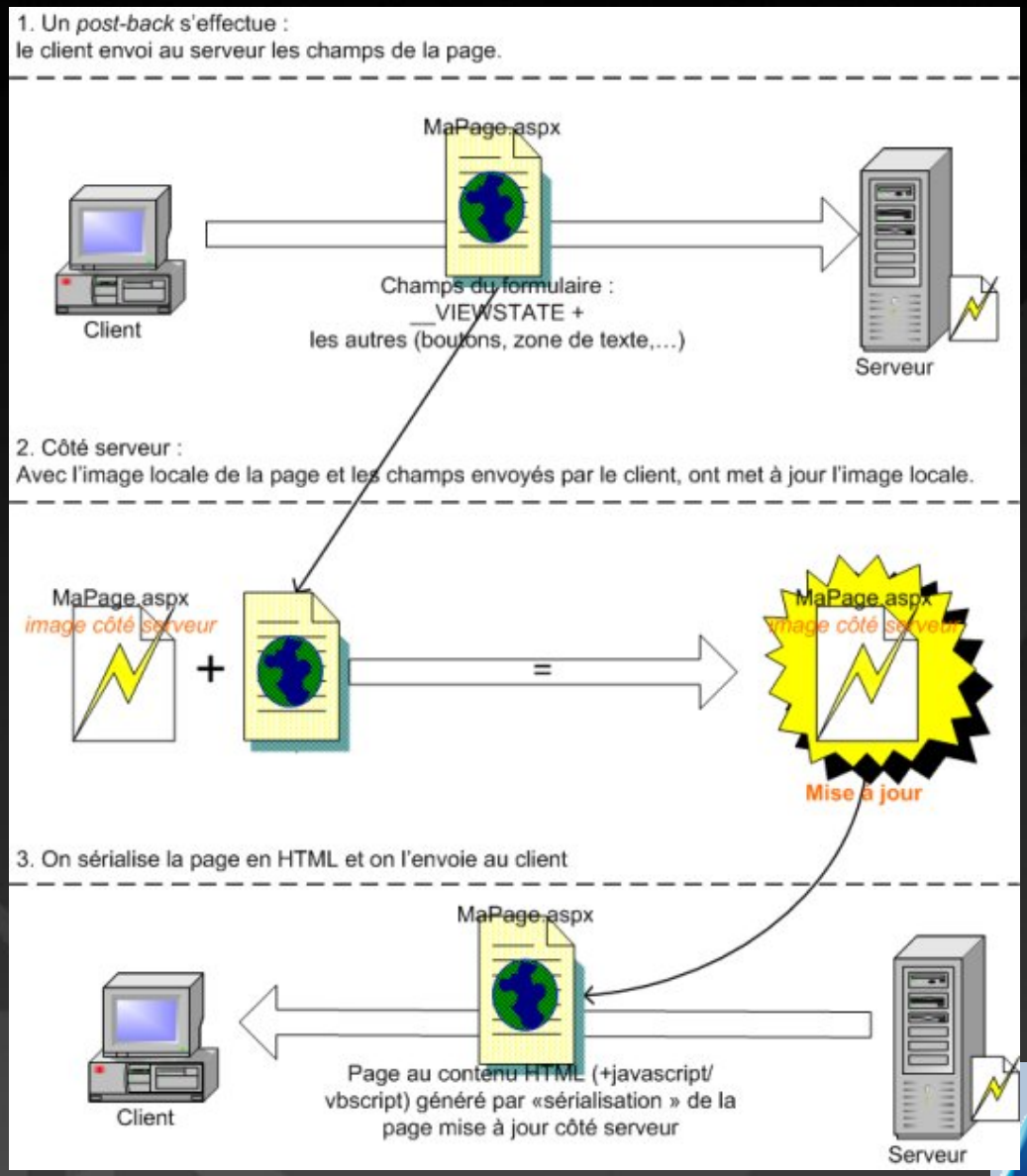
- Langage événementiel
- Page ASPX / Code beside
- Cycle de vie d'une page
- Viewstate

- ➔ Visual Studio

# ASP.NET


Serveur ASP.NET
Mapage.
aspx
1. Demande d'une page
Client
Mapage.
aspx
5. On met le résultat en
cache et on envoi la
version HTML au client
4. Le CLR compile
la page en MSIL
3. ASP.NET demande
la compilation par
le CLR
Cache
ASP.NET
(GAC)
2. Vérification dans
le cache ASP.NET
Mapage.aspx
Mapage
.aspx.cs

# ASP.NET


1. Un post-back s'effectue :
le client envoi au serveur les champs de la page.
MaPage.aspx
Client
Champs du formulaire :
__VIEWSTATE +
les autres (boutons, zone de texte,...)
Serveur
2. Côté serveur :
Avec l'image locale de la page et les champs envoyés par le client, ont met à jour l'image locale.
MaPage.aspx
image côté serveur
+
=
MaPage.aspx
image côté serveur
Mise à jour
3. On sérialise la page en HTML et on l'envoie au client
MaPage.aspx
Client
Page au contenu HTML (+javascript/
vbscript) généré par «sérialisation » de la
page mise à jour côté serveur
Serveur

# ASP.NET

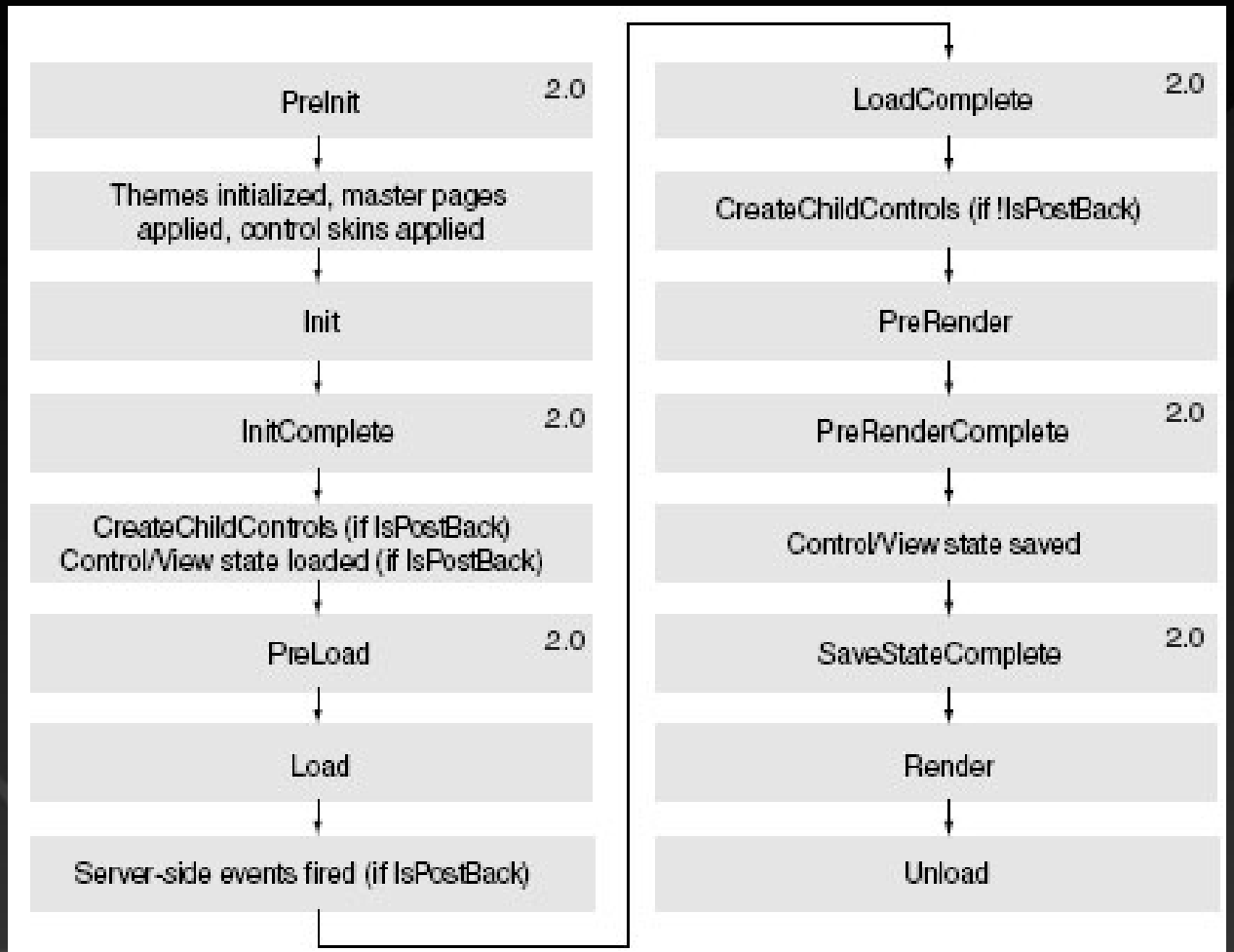



FIGURE 1-2: Events in the page lifecycle

# 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 3.0

# 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
  - Hastable
  - Dictionnary
  - SortedList
  - SortedDictionnary
  - Queue
  - Stack

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.Write("{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 sont définies par le framework mais vous pouvez en créer.

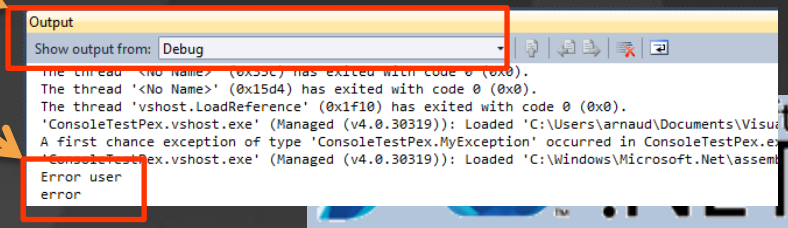
```
public class MyException : Exception
{
    public MyException(string message) : base(message)
    {
    }
}
```

```
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 peuvent ne pas être 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 :
  - Environnement.CurrentDirectory
  - Environnement.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 la gestion de mémoire. 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

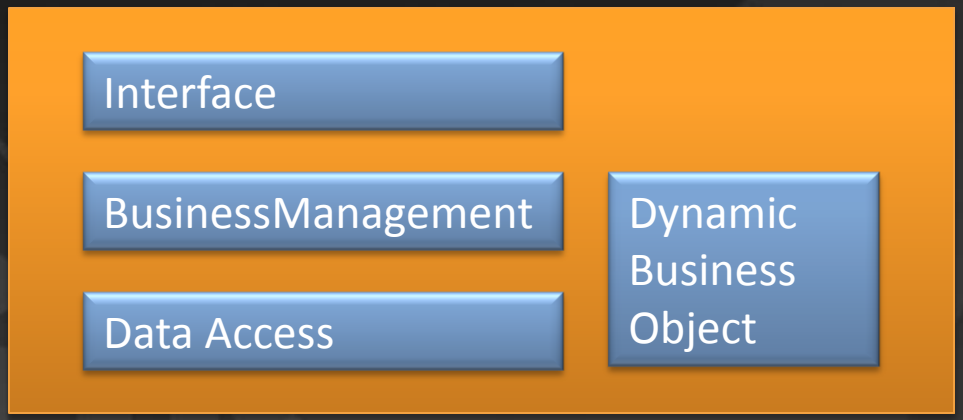
```
private static void DisplayNotNull<E>(E value)
{
   if (value != null)
      Console.WriteLine(value);
   else
      Console.WriteLine("null value");
}
```

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

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

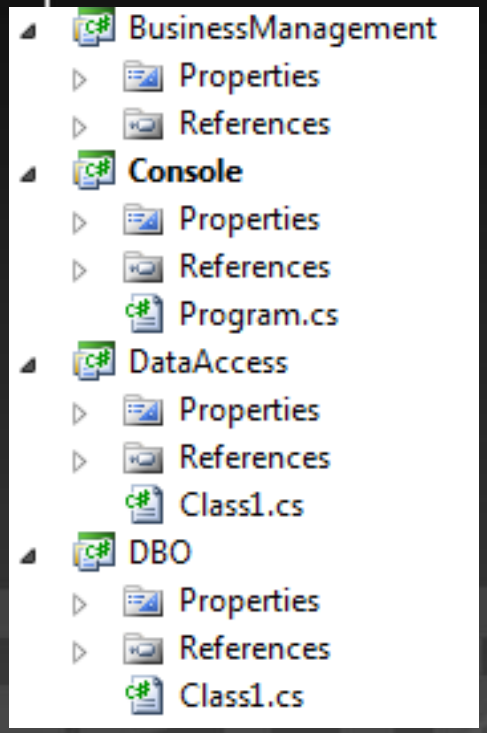
# Le langage c#

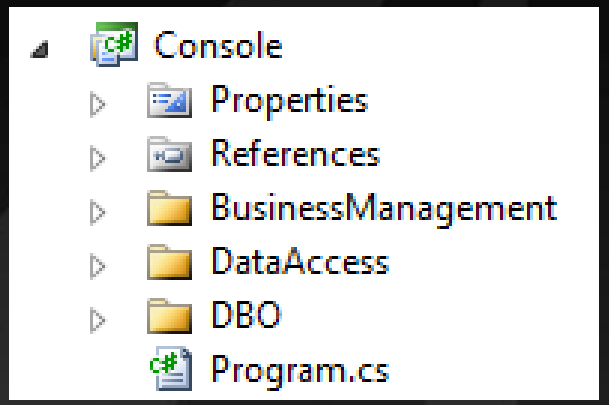
- Architecture n tier
  - DAL : Data Access Layer
  - BL : Business Layer
  - UI : User Interface
  - Entity layer (couche transversale)



# Le langage c#

- Et en concret ?
  - Soit créer les couches sous forme de dossiers
  - Soit sous forme de DLL

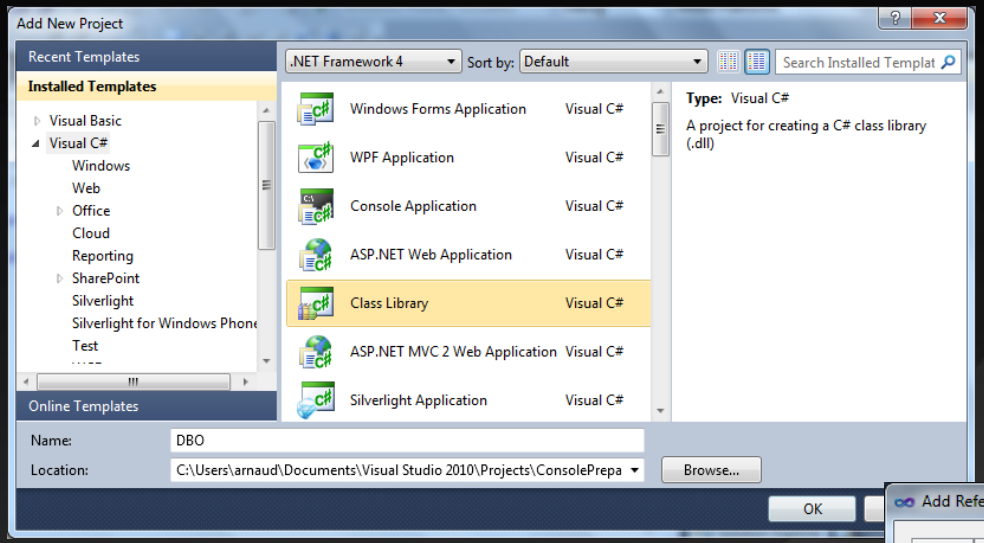


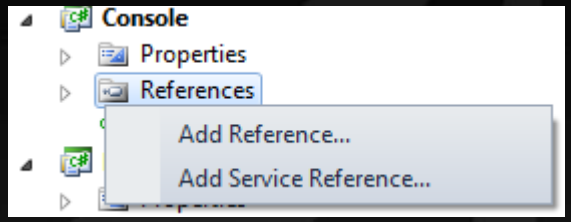




Comment choisir ?
Si vous devez réutiliser du code métier alors il faut passer par par une DLL. Par exemple : si vous devez utiliser des interfaces différentes (Application console et application web) mais avec le même code

# Le langage c#

- Comment créer une DLL ?
  Clic droit sur une solution -> add -> new project

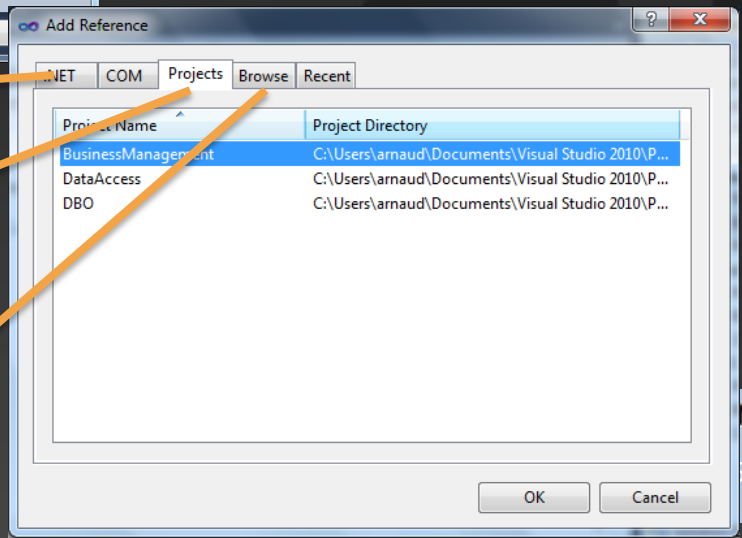



Assembly .NET

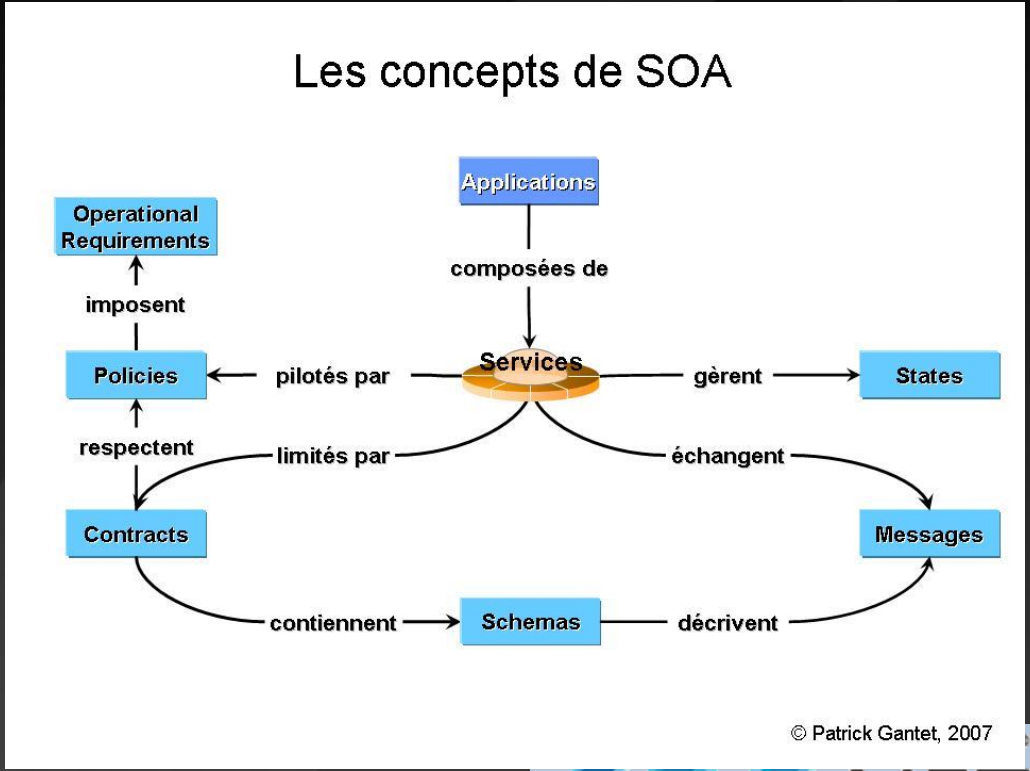
Projet dans la solution

DLL externe à la solution *.dll



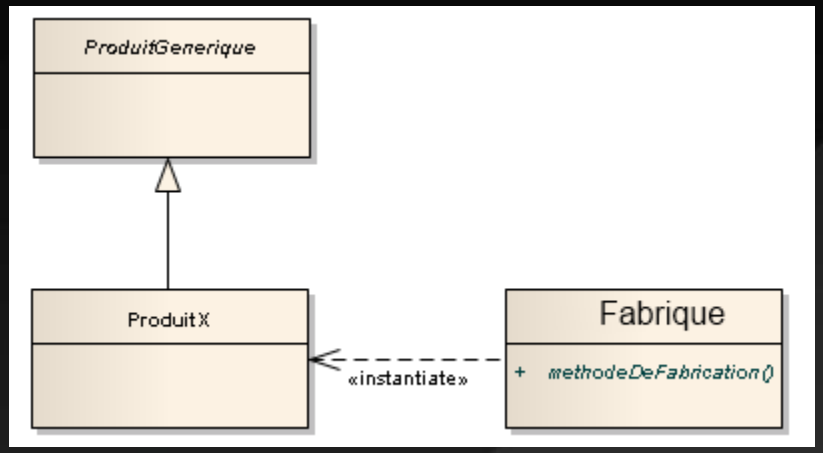
# LE LANGAGE C#

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



# 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();
        }
    }
}
```

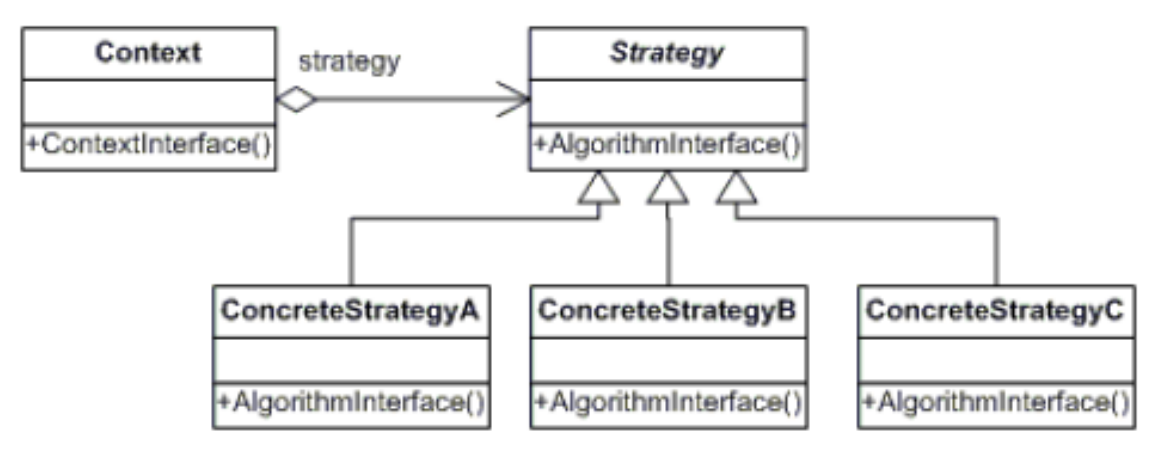
Implémente IProduit

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 champs 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;
    }
}
```

Microsoft .NET

# LE LANGAGE C#

Utilisation :

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

# Mise en Pratique


# Mise en Pratique

- Installation / 1er utilisation de visual studio 2010. 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.