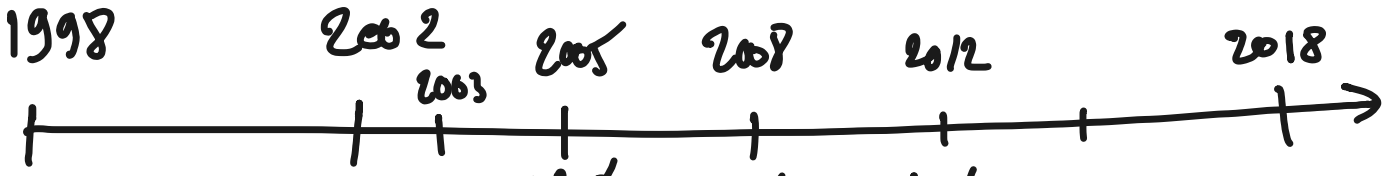


.net FW (1.0)

→ ASP.net (Web)

WPF (vista)

dernière Visual Studio
→ VB
→ C++

Code natif.

.net Full (Standard)

.net Core → open Source

ASP.net MVC, Winform } .net standard ("full power")

ADO.net → Données

ASP.net → Web

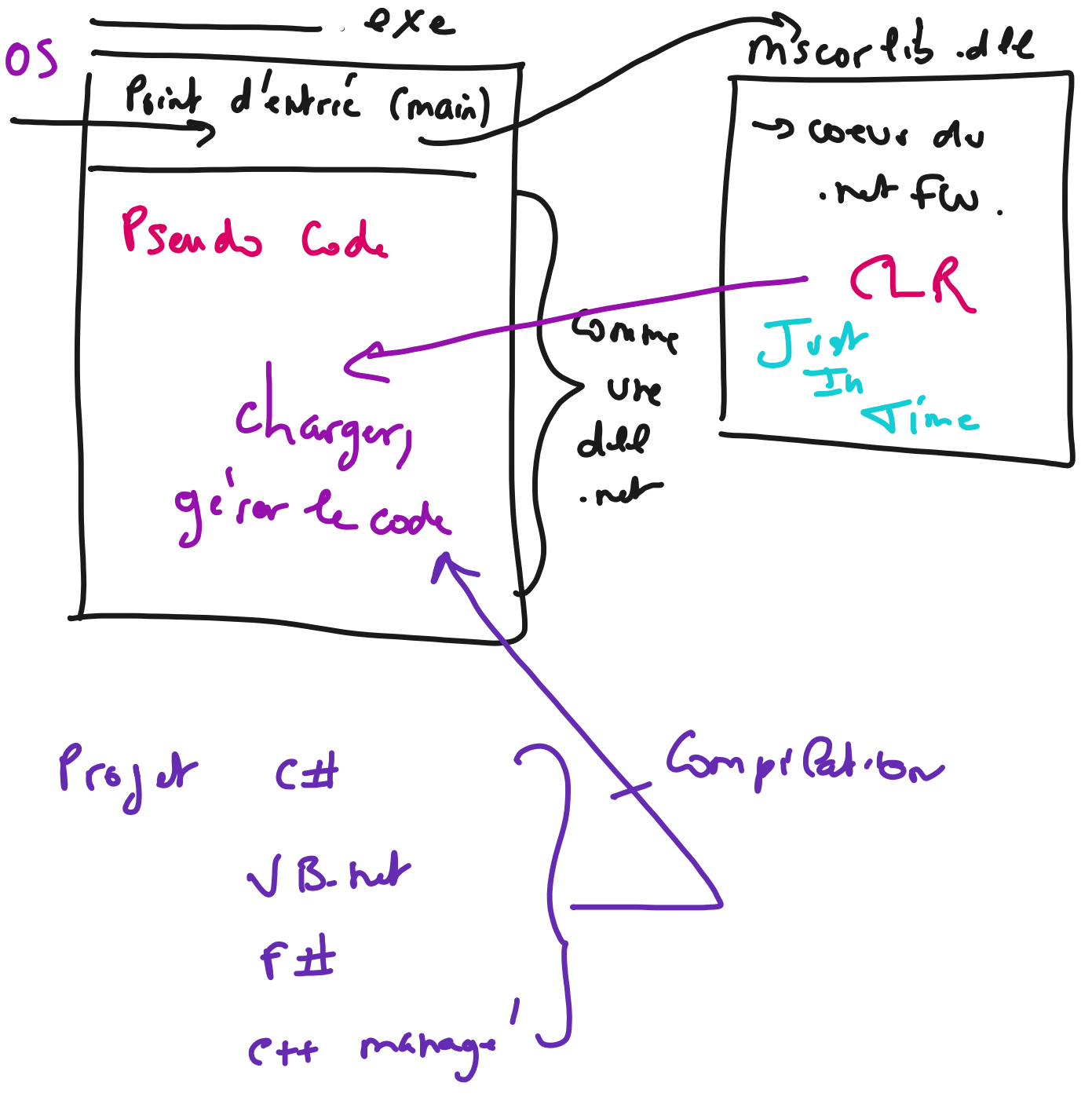
Winform → client lourd

Console

ASP.net MVC core
→ Linux
→ pas toutes les fonctionnalités

WPF → evolution de Winform

actuellement → Universal Window platform (evol. de WPF)


OS
.exe
Point d'entrée (main)
Pseudo Code
mscorlib.dll
→ coeur du .net FW.
CLR
Just In Time
Comme une dll .net
charger, gérer le code
Projet
C#
VB.net
F#
C++ managé
Compilation

1 Visual Studio

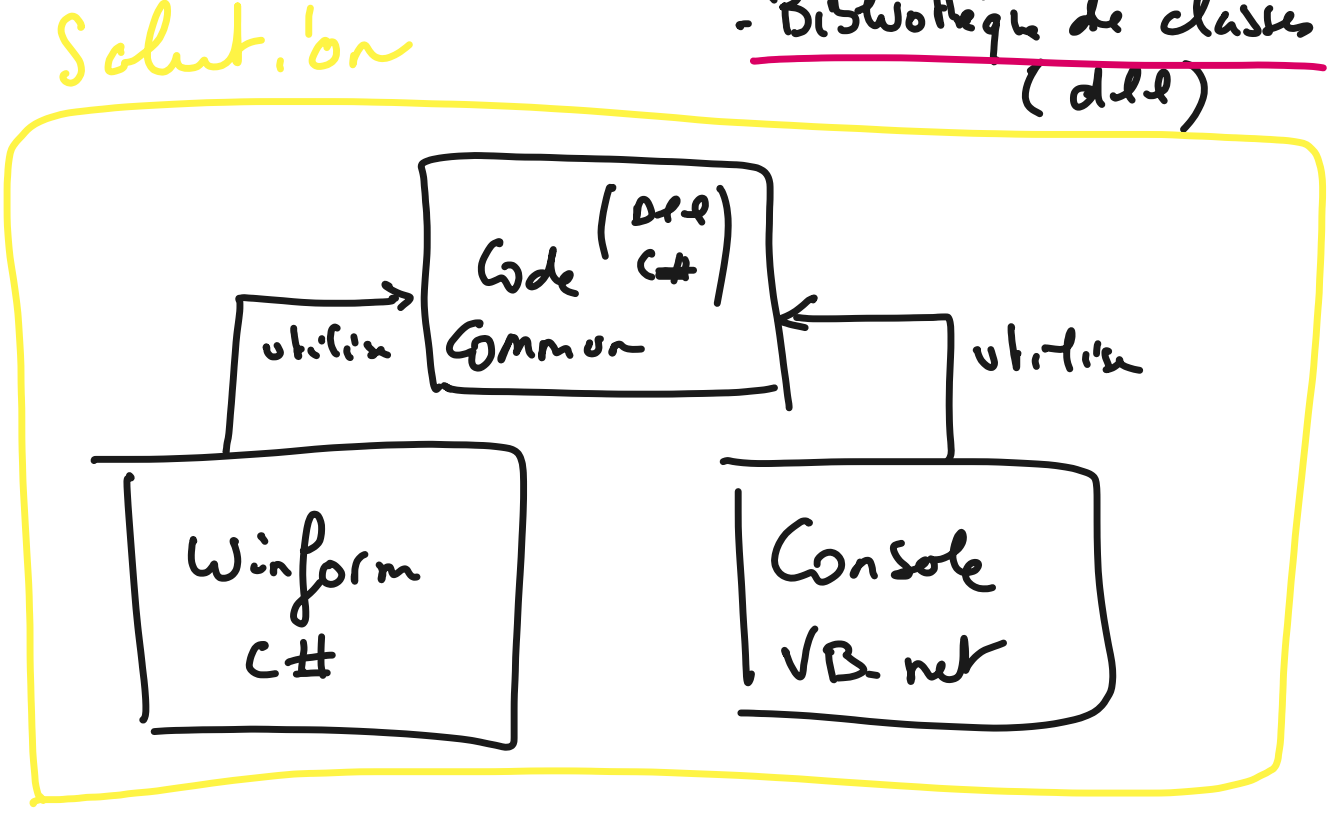
↳ 0..1 Solution chargée
(fichier .sln)
→ une liste de Projets

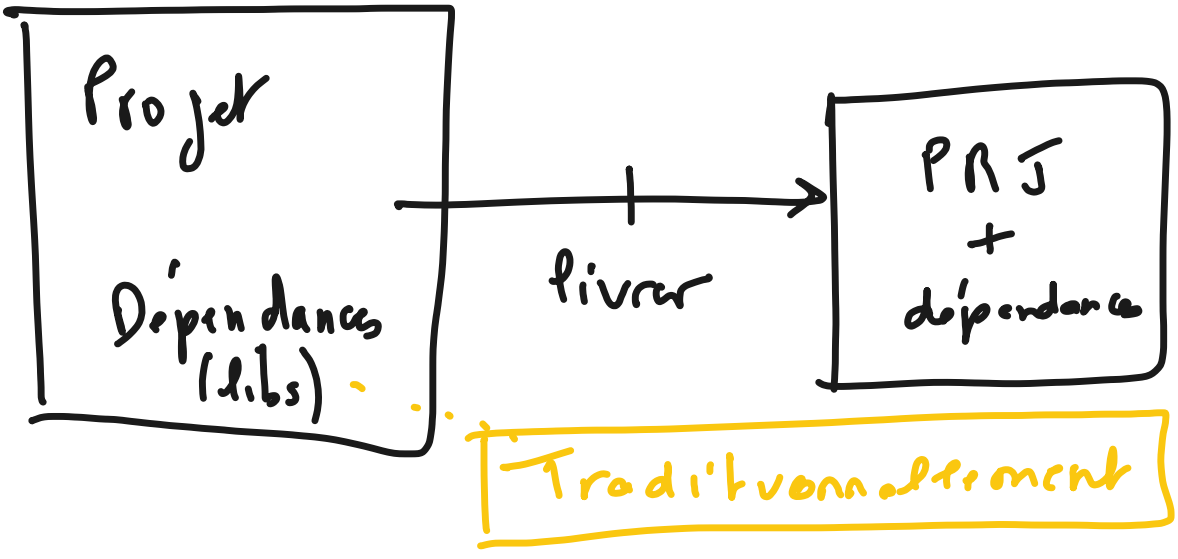
↳ 0..* Projets

1 Projet = 1 Langage

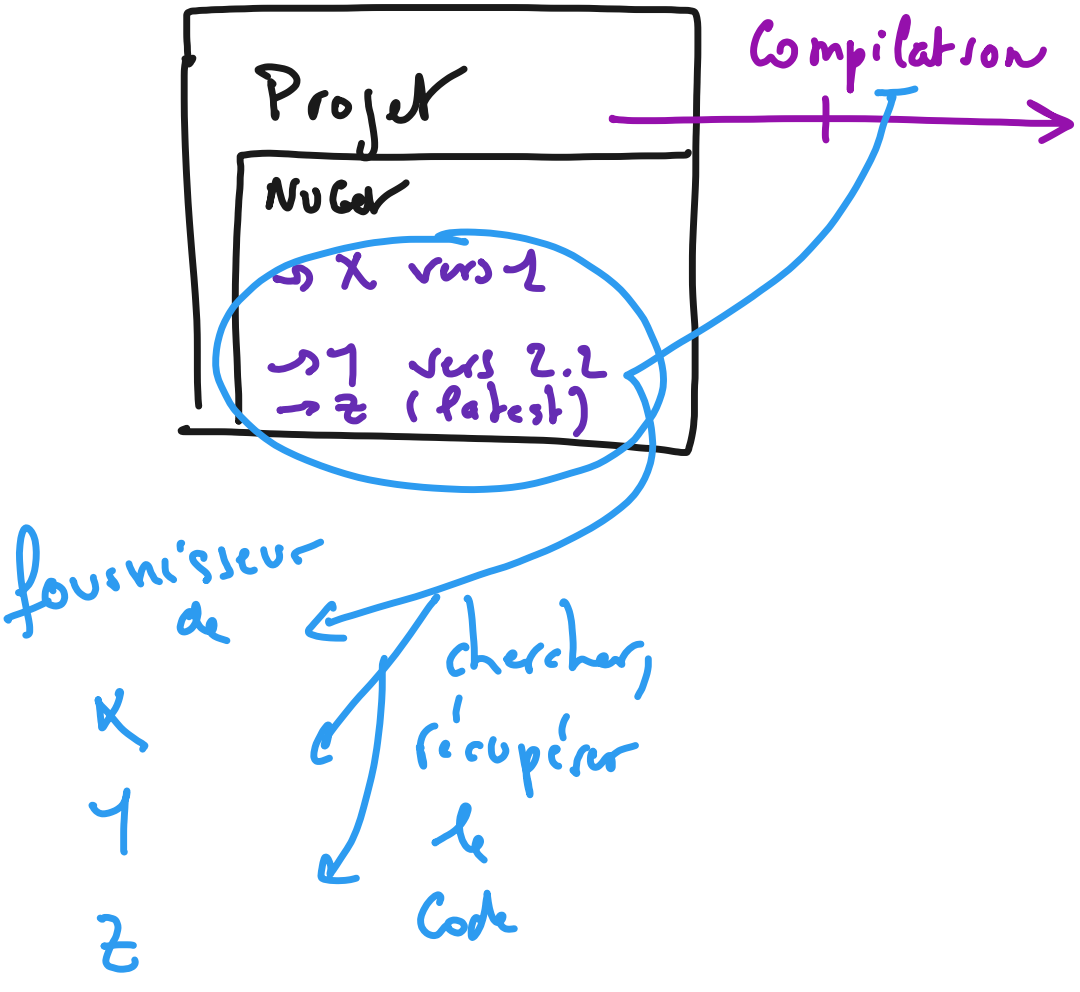
- C#
- VB.net
- F#
- C++ managed
- ...

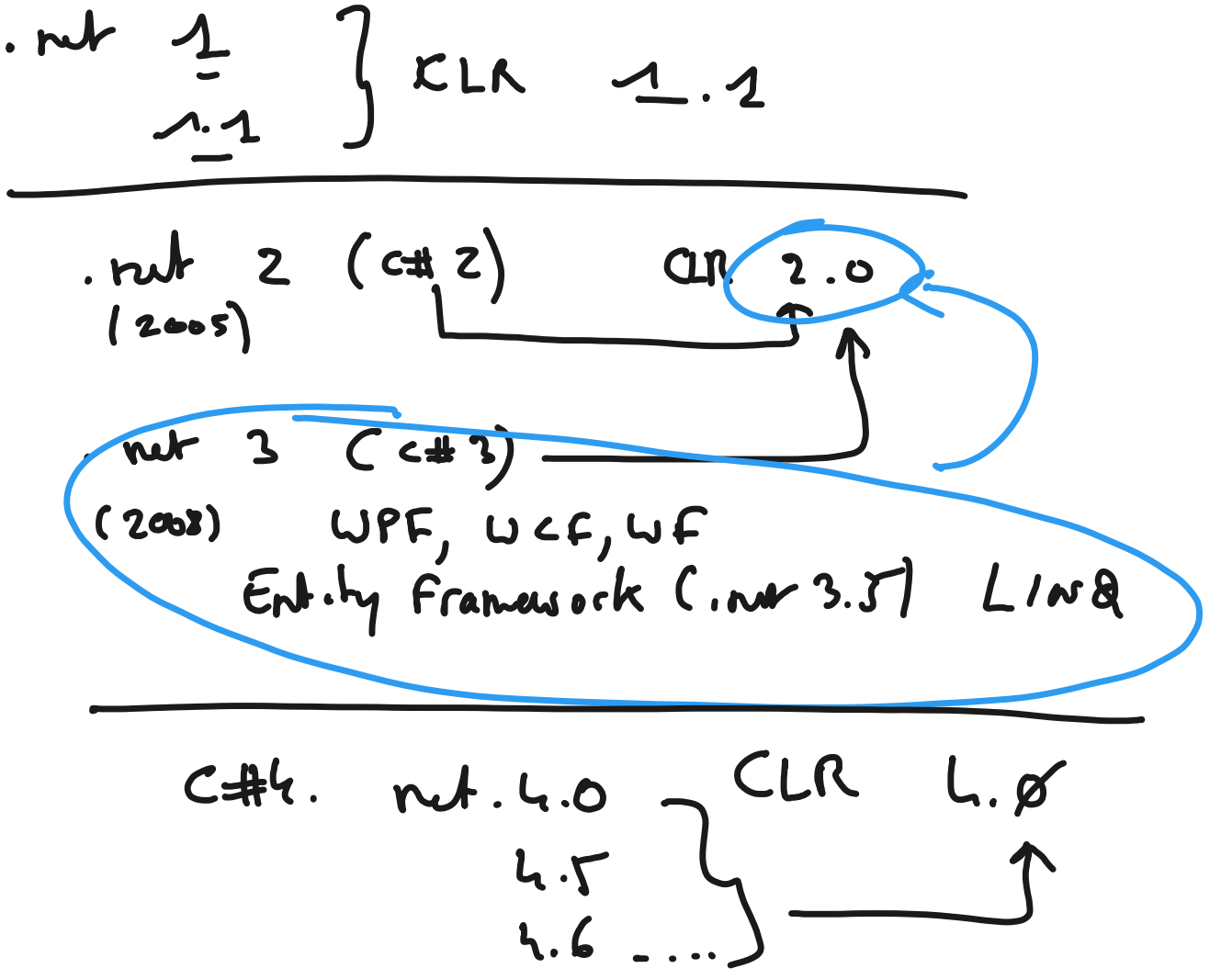
type de sortie:
- Console
- WinForm
- WPF
- UWP
- ASP.net
- Bibliothèque de classes (dll)





Modèle de Packages




.net 1
1.1
CLR 1.1
.net 2 (c# 2)
(2005)
CLR 2.0
.net 3 (c# 3)
(2008)
WPF, WCF, WF
Entity Framework (.net 3.5) LINQ
c#4. .net 4.0
4.5
4.6 ....
CLR 4.0

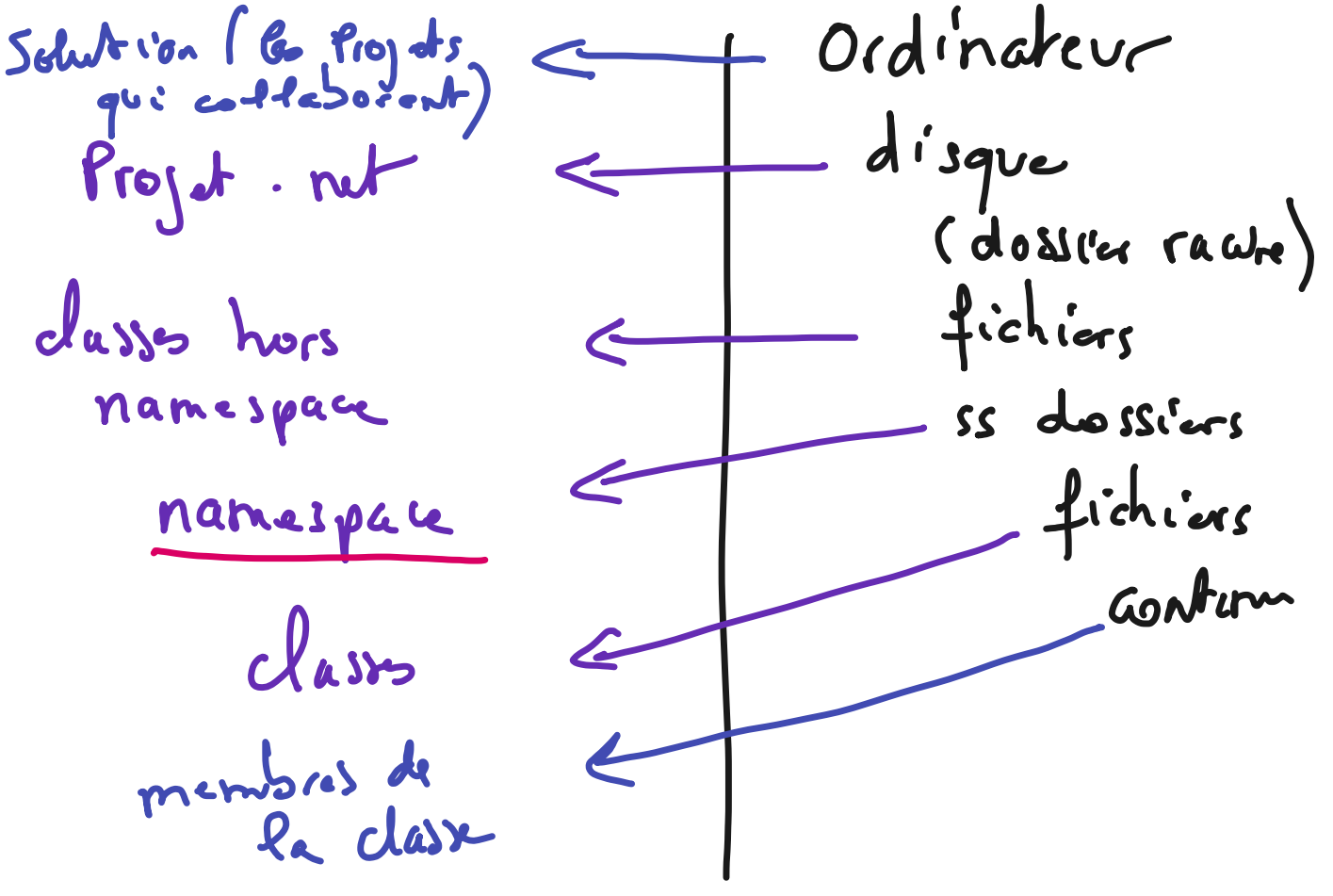
Structuré

.cpp ← module (fichier de code)

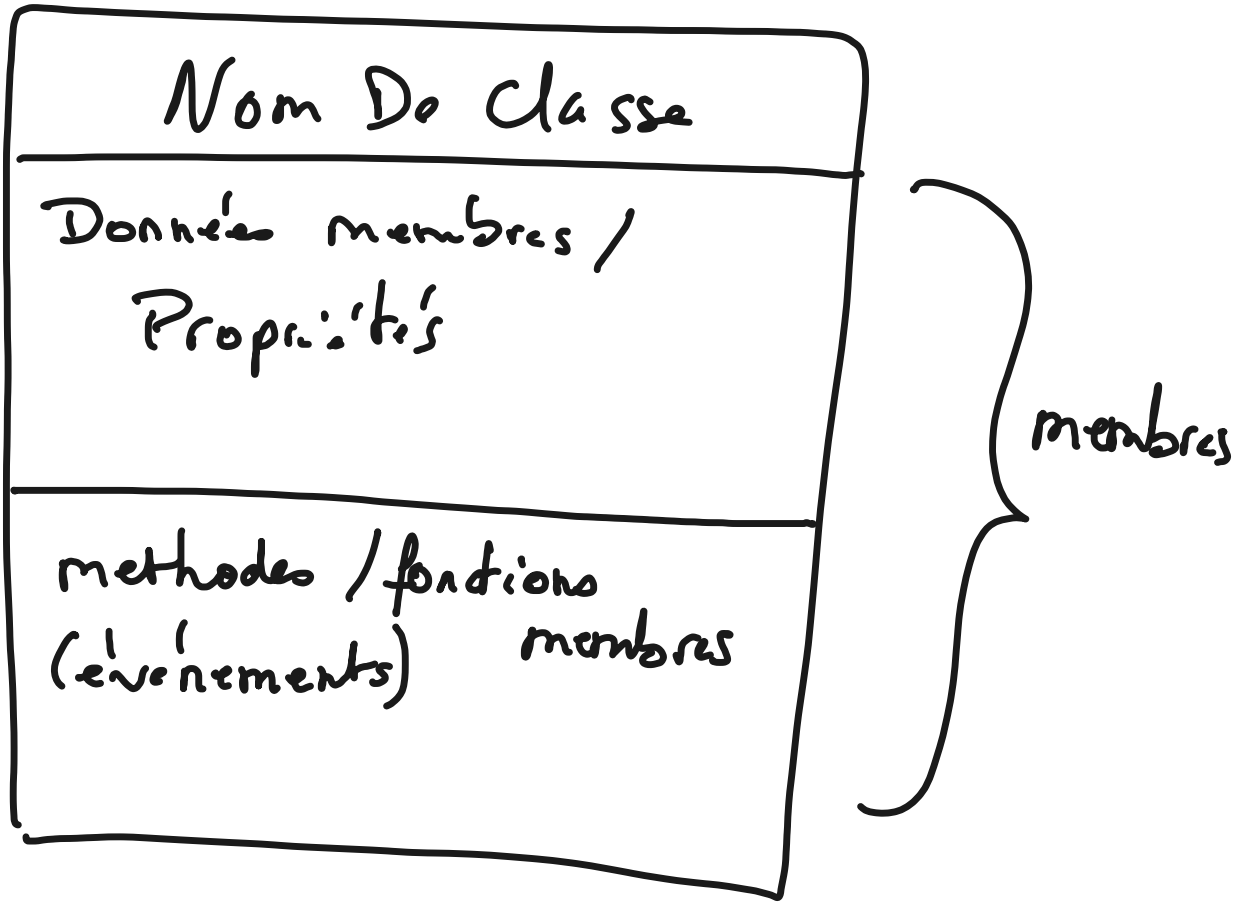
```
fonction ()
{ code
```

→ Tout est dans des classes

→ Milliers de classes.



UML


Nom De Classe
Données membres / Propriétés
méthodes / fonctions (événements)
membres
membres

Conventions

Pascal → nom Propre

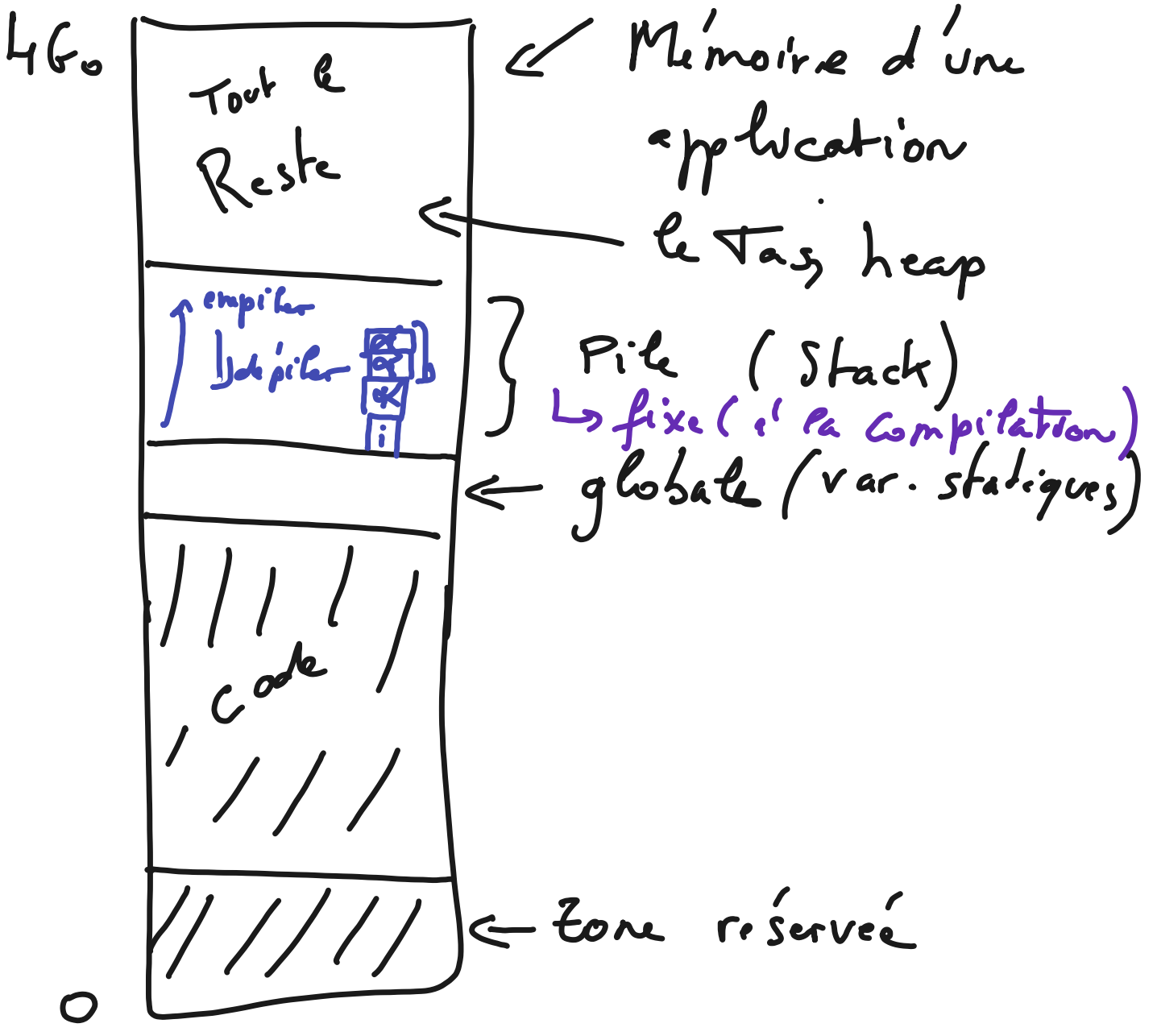
lowercasing

UPPERCASING

PascalCasing

camelCasing

NameSpace	:	PC
Types (classes)	:	PC
Methodes	:	PC
Données membres	:	PC
Paramètres	:	cC
variable (locale)	:	lc


4Go
Tout le Reste
Mémoire d'une application
le tas heap
empiler
dépiler
i
Pile (Stack)
↳ fixe (à la compilation)
globale (var. statiques)
Code
Zone réservée
0

{

③ faire la relation

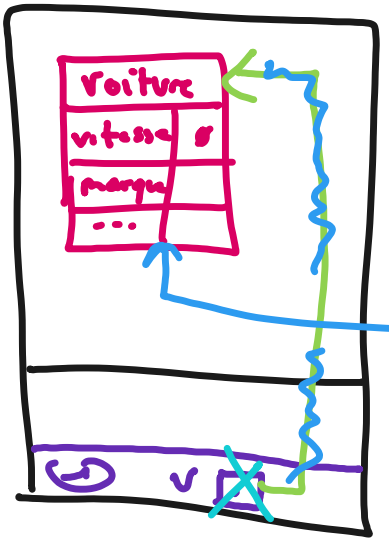
Voiture v = new Voiture();

② allouer v sur la pile

① v est une référence vers l'objet (un Pointeur)

}

④ Allouer la taille de champs de Voiture



Tas

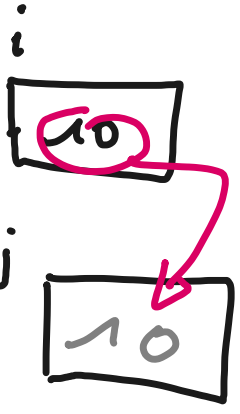
non référençable

Pile à nettoyer

Type simple / Valeur

int, double, char, ...

```
int i=10;
int j=i;
```



Par Copie

Complexes / Objets

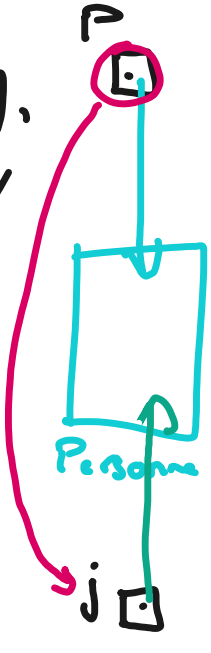
```
Personne p;
P= new Personne();
Personne j =p;
```

j. ....

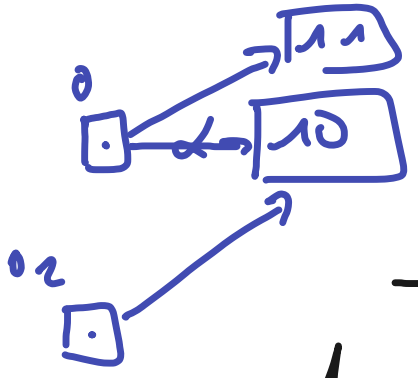
ou

p. ....

⇒ la même chose



Par Reference

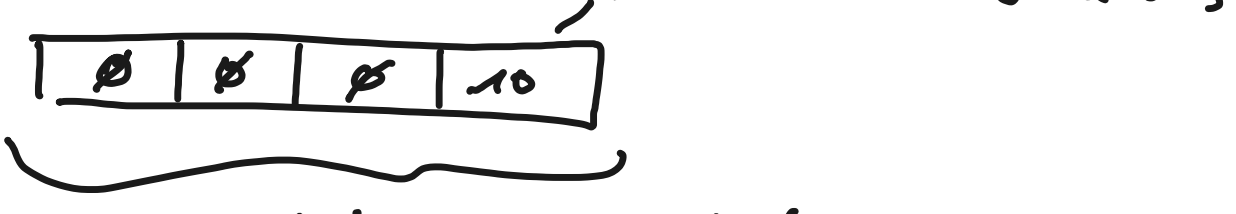


o = 11

→ Pour pouvoir dupliquer un objet, la bonne pratique consiste à implémenter l'interface ☰ Cloneable et sa méthode clone()

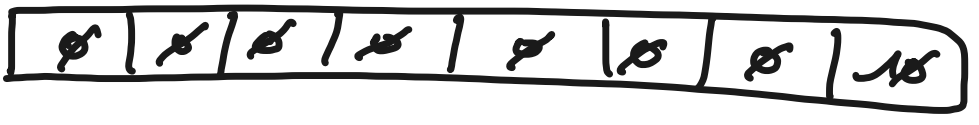
Transtypage = Voir la même valeur sous un autre angle

Conversion = Transformer une valeur en une autre.

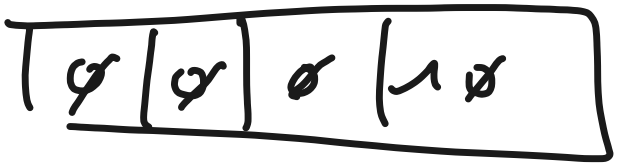


4 octets de 10

sous forme de long



10 numérique (32 bits) :



10.toString()
ou
Convert.ToString(10)

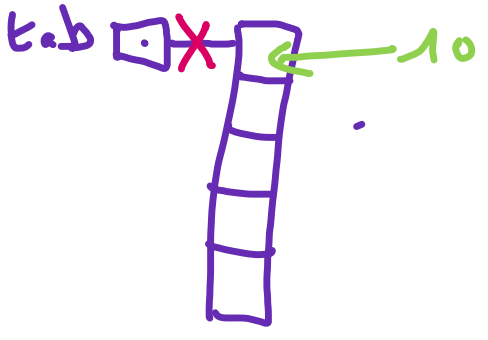
} renvoie un string

représentation mémoire

Conversion

'1' '0'

read only int[] tab = new int[5];



tab[0] = 10; ✓ ok

tab = ~~new~~ int[5];

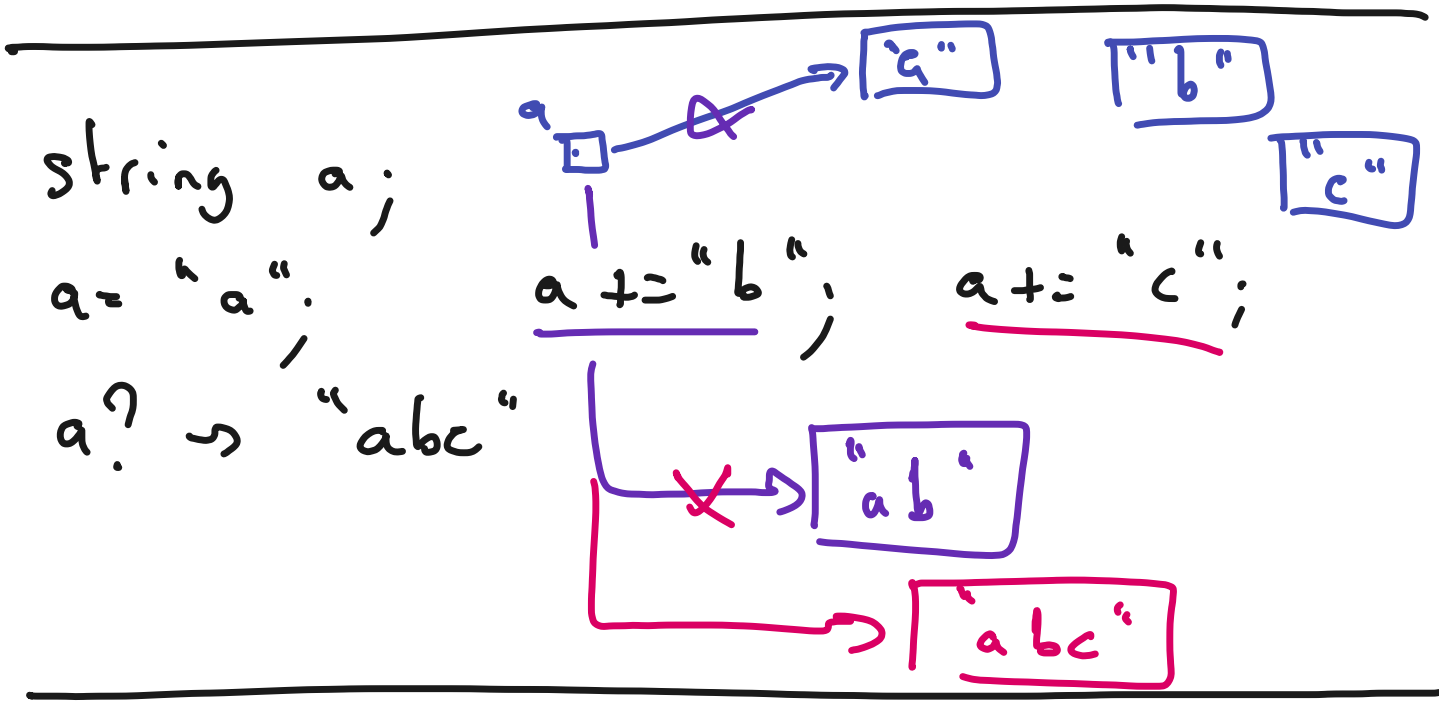
10 & 2 = 2 : 0010 (2) & 1010 (10) = 0010

10 | 2 = 10 : 0110 (6) | 0010 (2) = 0110 (10)

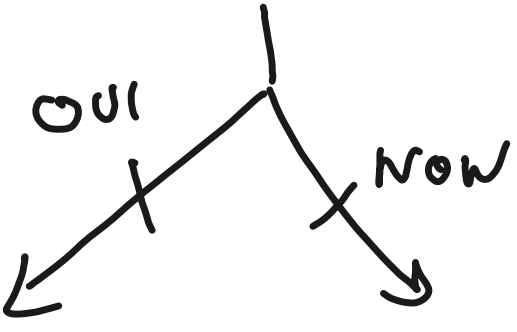
0110
& 0010
0010 → 2

128 64 32 16 8 4 2 1

1 0 1 0



est ce que je connais
à l'avance le nombre
d'éléments à stocker?

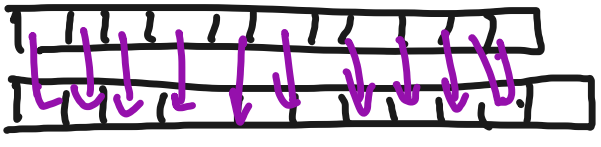


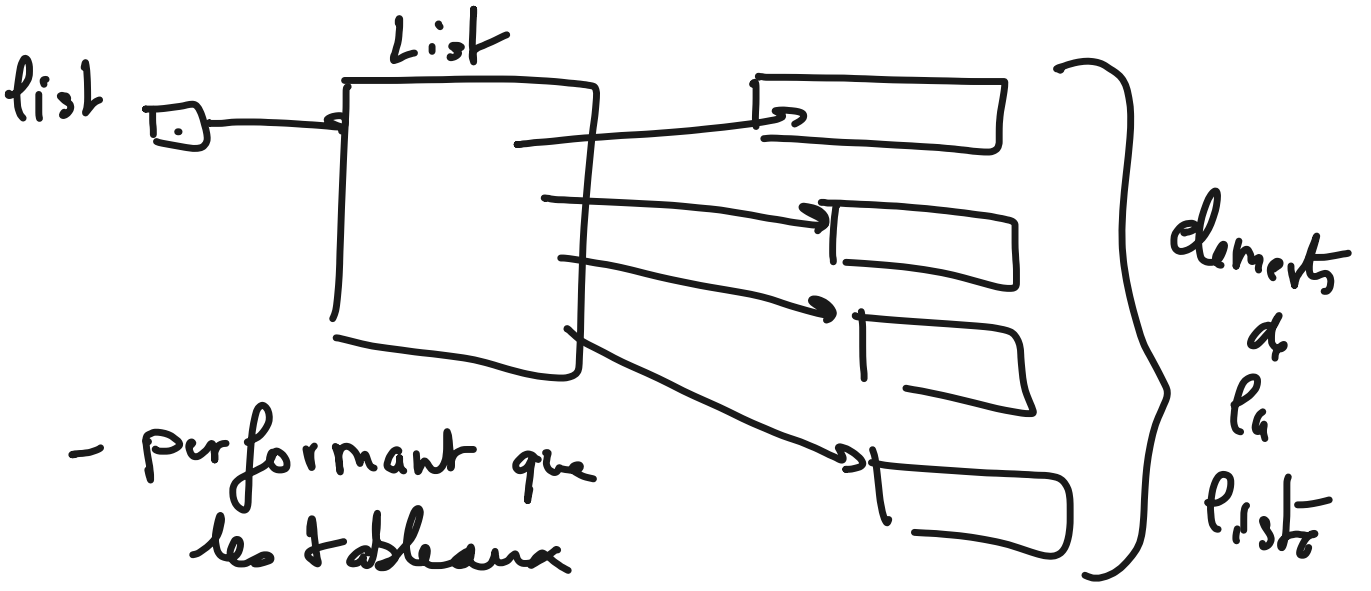
Tableau

Collection

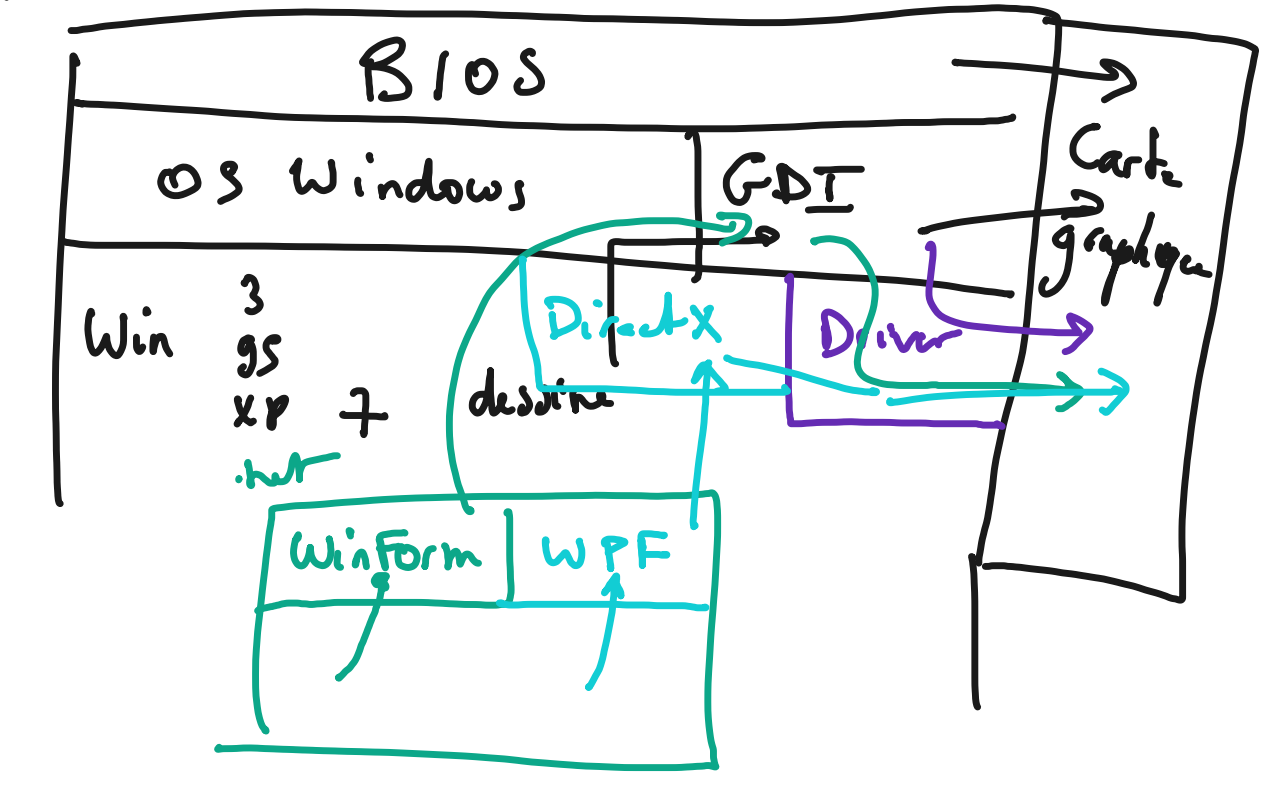
dim tab(10)

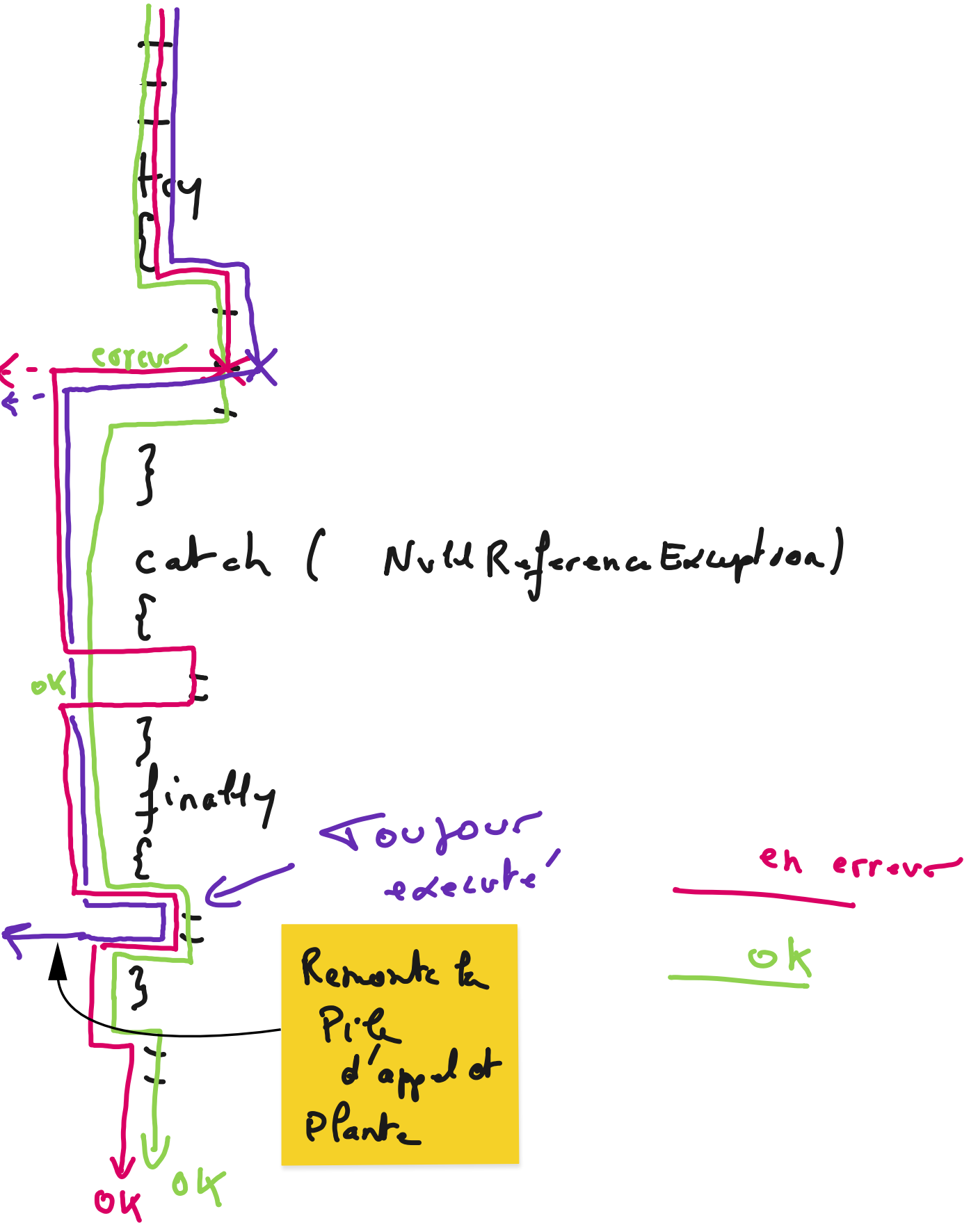
VB redim tab(11) Preserve





- performant que les tableaux

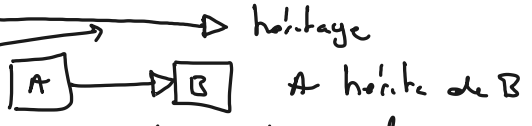



try
{
erreur
}
catch (NullReferenceException)
{
ok
}
finally
{
Toujours exécuté
}
Remonte la Pile d'appel et Plante
en erreur
ok
ok
ok

```
class PersonneException : Exception
   {
      public PersonneException(string msg) : base(msg)
      {
      }
   }

   class AgeIncorrectException : PersonneException
   {
      public AgeIncorrectException(string msg) : base(msg)
      {
      }
   }

   class Personne
   {
      private int age;    // l'age doit être entre 0 et 100

      public void setAge( int age)
      {
         if (age >= 0 && age < 101)
            this.age = age;
         else
            throw new AgeIncorrectException("Erreur d'age incorrect : " + age);
      }
      public int getAge()
      {
         return age;
      }
   }

   class Program
   {

      static void Main(string[] args)
      {
         Personne p = new Personne();

         bool bok = false;
         do
            try
            {
               p.setAge( Convert.ToInt32(Console.ReadLine()));
               bok = true;
            }
            catch (AgeIncorrectException ex)
            {
               Console.WriteLine("Il y a eu une erreur : " + ex.Message + " " +
ex.GetType().FullName);
            }

            catch (PersonneException ex)
            {
               Console.WriteLine("Il y a eu une erreur : " + ex.Message + " " +
ex.GetType().FullName);
            }

            catch (Exception ex)
            {
               Console.WriteLine("Il y a eu une erreur : " + ex.Message + " " +
ex.GetType().FullName);
            }

         while (!bok);


         Console.WriteLine(p.getAge());
      }

   }
```

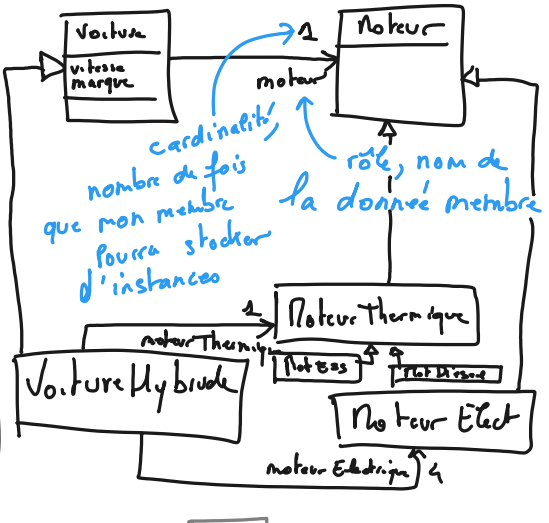
UML : Diagramme de Classes

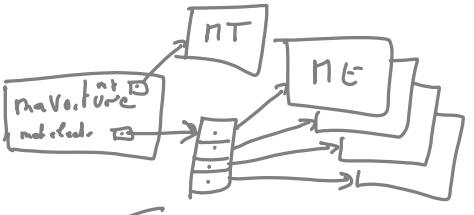
- - - - - : Utilisation
d'un doit pouvoir voir d'autre (simple)



relation "est-un" Etre

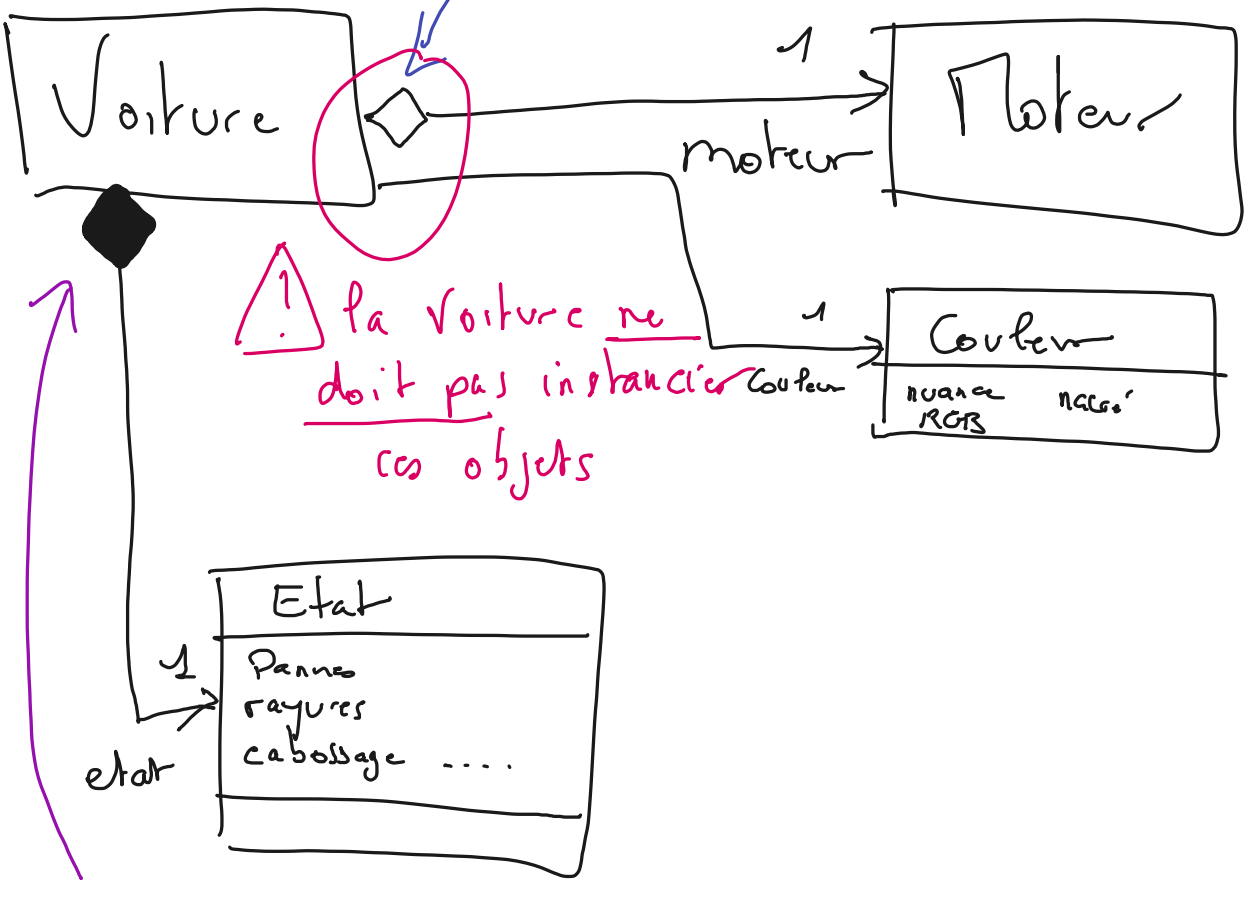
Aggrégation (relation "a-un" (avoir))





```
namespace ConsoleApplication1
{
   class Vehicule
   {

   }
   class Moteur
   {
   }
   class Voiture : Vehicule
   {
      public Moteur moteur;
      public string marque, modele;

      public Voiture()
      {
      }
   }

   class VehiculeFactory
   {
      private static VehiculeFactory vf;
      public int nbVoiture;

      // Pattern Factory
      public Voiture createVoiture( string marque, string modele )
      {
         Voiture tmp = new Voiture();
         tmp.moteur = new Moteur();
         tmp.marque = marque;
         tmp.modele = modele;

         nbVoiture++;

         return tmp;
      }


      // Pattern SingToN
      public static VehiculeFactory getFactory()
      {
         if (vf == null)
            vf = new VehiculeFactory();
         return vf;
      }
   }



   class Program
   {
      static void Main(string[] args)
      {
         Voiture v = VehiculeFactory.getFactory().createVoiture("Dodge", "Viper");
         Console.WriteLine(VehiculeFactory.getFactory().nbVoiture);
      }
   }

}
```

losange = plein juste informatif de relation de contenant / contenu



⚠ la voiture ne doit pas instancier ces objets Couleur

Composition, ou "Aggregation forte" est possible si le cycle de vie est identique et non interchangeable en instance d'hôtes

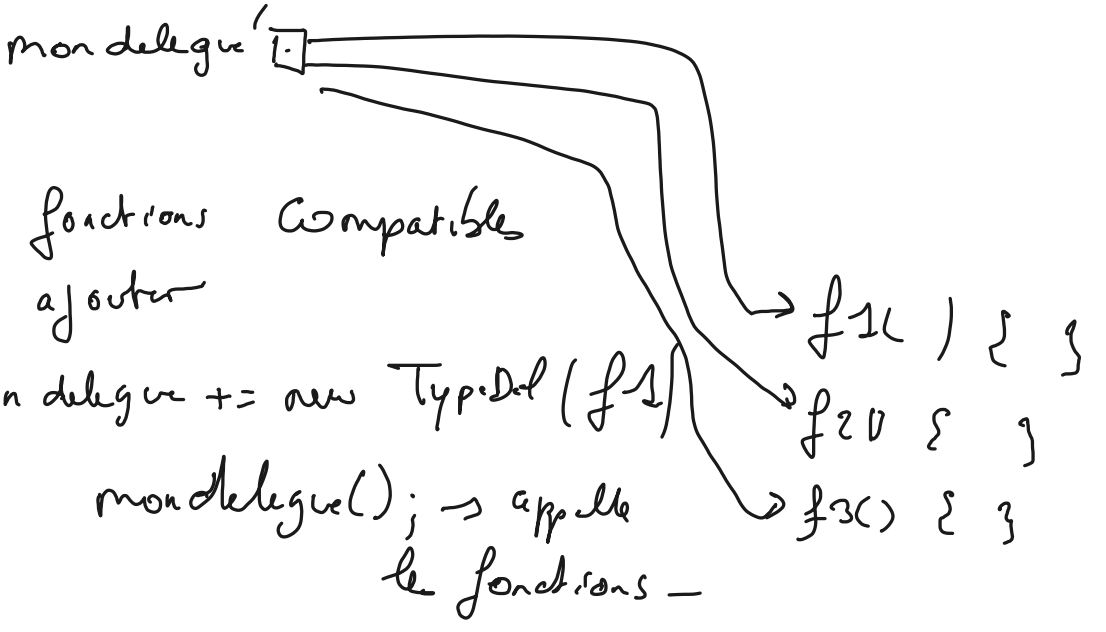
⇒ Seulement dans ce cas de figure, la classe hôte instanciera le contenu -

```
namespace ConsoleApplication2
{
   interface IDescriptible
   {
      string SeDecrire();
   }
   class Cercle : IDescriptible, ICloneable
   {
      public int Rayon { get; set; }

      object ICloneable.Clone()
      {
         return new Cercle() { Rayon = this.Rayon };
      }

      string IDescriptible.SeDecrire()
      {
         return "Je suis un cercle de Rayon " + Rayon;
      }
   }
   abstract class Animal : IDescriptible
   {
      public abstract string Communiquer();
      // Je dois retransmettre explicitement les méthodes abstraites d'interface
      public abstract string SeDecrire();
   }
   class Chien : Animal
   {
      public override string Communiquer()
      {
         return "Waffff";
      }

      public override string SeDecrire()
      {
         return "Wafff";
      }

   }
   class Voiture : IDescriptible
   {
      public string SeDecrire()
      {
         return "Vrooom";
      }
   }

   class Program
   {
      static void Main(string[] args)
      {
         List<Object> objs = new List<Object>();
         objs.Add(new Cercle());
         objs.Add(new Chien());
         objs.Add(new Voiture());
         foreach (IDescriptible o in objs)
            Console.WriteLine("L'objet est : "+o.SeDecrire());
      }
   }
}
```

Délégués

1) Définition le type des methodes pointées
avec le nom du type de délégué.

2) Déclarer une variable de ce type



3) declarer fonctions compatibles
et les ajouter
mondelegue += new TypeDel(f1);

4) appeler mondelegue(); → appelle
les fonctions.

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

namespace ConsoleApp2
{
    class Program
    {
        class UsineEventArgs : EventArgs
        {
            public Usine Usine { get; set; }
        }
        abstract class Pompe
        {
            public abstract void Refroidir(object sender, UsineEventArgs e);
        }

        class PompeHydraulique : Pompe
        {
            public override void Refroidir(object sender, UsineEventArgs e)
            {
                e.Usine.Temperature = e.Usine.Temperature - 4;
            }
        }
        class PompeElectrique : Pompe
        {
            public override void Refroidir(object sender, UsineEventArgs e)
            {
                e.Usine.Temperature = e.Usine.Temperature - 2;
            }
        }

        class Usine
        {
            // Déclarer un type de délégué : représente la forme des methodes appelables
            // le nom du délégué deviens un type ( est une classe ) : SurchauffeDelegate
            public delegate void SurchauffeDelegate(object sender, UsineEventArgs e);

            // Déclarer la variable qui servira d'évènement :
            public event SurchauffeDelegate OnSurchauffe { get; set; }

            public int Temperature { get; set; }
            public void Produire()
            {
                Temperature = Temperature + 10;
                if (Temperature > 80)
                {
                    Console.WriteLine("Surchauffe");
                    if (OnSurchauffe != null) // Toujours vérifier qu'il y a au moins un branchement
                        OnSurchauffe(this, new UsineEventArgs() { Usine = this } );
                }
            }
        }

        // Déclarer un type de délégué pour l'expression Lambda :
        delegate int CalculDelegate(int i);

        static void Main(string[] args)
        {
            Usine u = new Usine();
            Pompe p = new PompeHydraulique();
            Pompe p2 = new PompeHydraulique();
            Pompe p3 = new PompeElectrique();
            Pompe p4 = new PompeElectrique();

            // A : méthode d'origine de déclaration de méthode externe
            // Je branche la méthode Refroidir de ma pompe :
            u.OnSurchauffe += new Usine.SurchauffeDelegate( p.Refroidir);

            //u.OnSurchauffe += new Usine.SurchauffeDelegate(p2.Refroidir);
            //u.OnSurchauffe += new Usine.SurchauffeDelegate(p3.Refroidir);
            //u.OnSurchauffe += new Usine.SurchauffeDelegate(p4.Refroidir);

            // B : Déclaration via une méthode anonyme :
            // le mot clef delegate ici sert a déclarer une méthode anonyme :s
            u.OnSurchauffe += new Usine.SurchauffeDelegate(delegate(object sender,
UsineEventArgs e)
                {
                    e.Usine.Temperature = e.Usine.Temperature - 1;
                });

            // C : Simplifie encore plus via les Lambda :
            // ( paramètres a passer )   =>  { code }
            u.OnSurchauffe += new Usine.SurchauffeDelegate((sender, e) =>
                {
                    e.Usine.Temperature = e.Usine.Temperature - 1;
                });

            // Expression Lambda : simplification de la méthode Lambda :
            // je déclarer anonymement le code de ma fonction de calcul ( c'est un exemple )
            // l'expression doit renvoyer une valeur, qui sera implicitement dans le return
            CalculDelegate cd  = (i) => i + 1;
            // Equivalent en methode Lambda :
            CalculDelegate cd2 = (i) => { return i + 1; };

            // Exemple final d'usage :
            int[] tabi = new int[10];
            tabi[2] = 2;
            tabi[3] = 3;
            tabi[8] = 3;
            var res = tabi.Where(e => e >= 2 && e <=5 );
            foreach (var r in res)
                Console.WriteLine("Résultat : "+r);

            for (; ; )
            {
                u.Produire();
                System.Threading.Thread.Sleep(500);
                Console.WriteLine("Température : "+u.Temperature);
            }
        }
    }
}
```

Persistance des données

1) Flux et Reader/Writers → la base de tout (interviews)

2) objet (Complexe) ⟷ Flux (fichier, ...) (assez commun)

Sérialisation / désérialisation

↳ Sérializable

3) XML → DOM (Document object Model)

XDocument, XElement, XAttribute

→ doc.save("_____.xml")

4) ADO.net ⇒ Méthodes/objets de base pour attaquer les BDD

→ Requêtes Sql, connecté/ Déconnecté

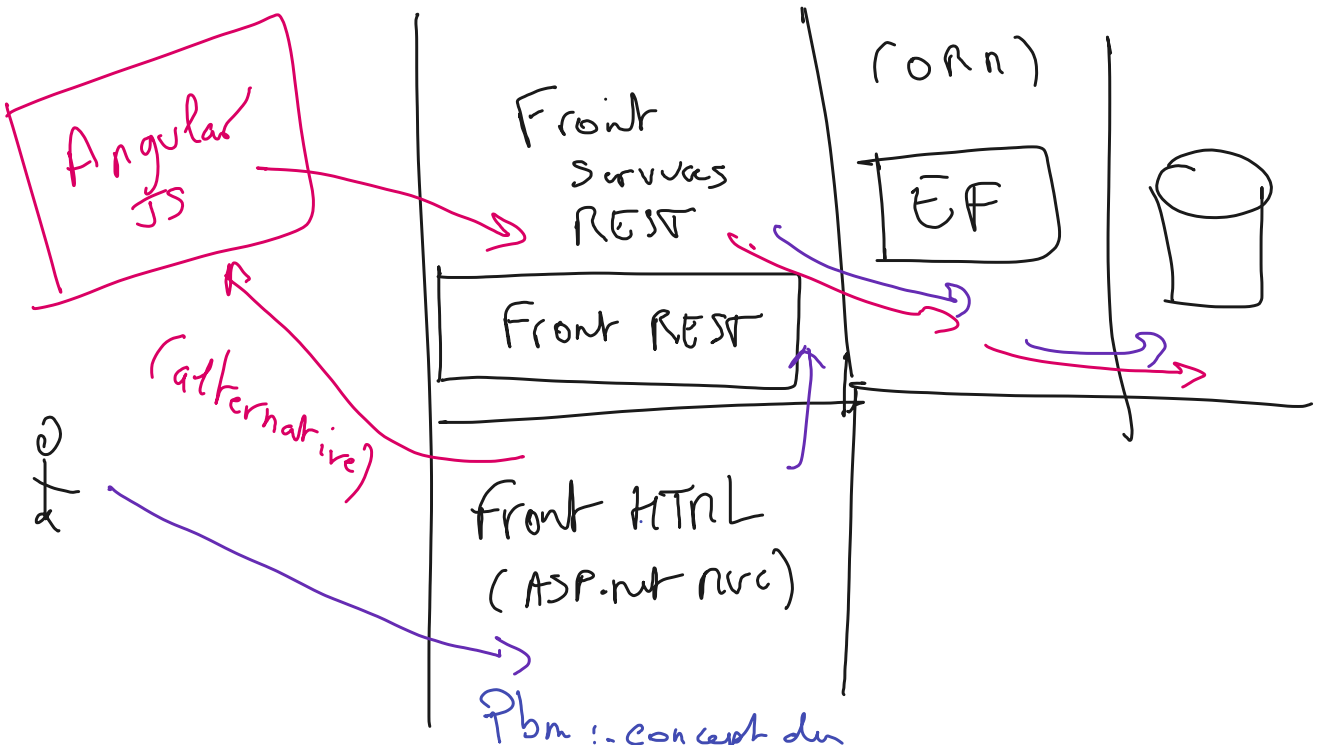
(5) Top = Entity Framework

⇒ Implémentation du Design Pattern ORM (Object-Relational Mapping)

Pour du boulot: nouveau d'architecture

Conseillé: ASP.net MVC + Entity Framework

Architecture actuelle.



Pbm : - concept du MVC à comprendre
- beaucoup de "Magie"
→ beaucoup de choses se passent derrière la scène