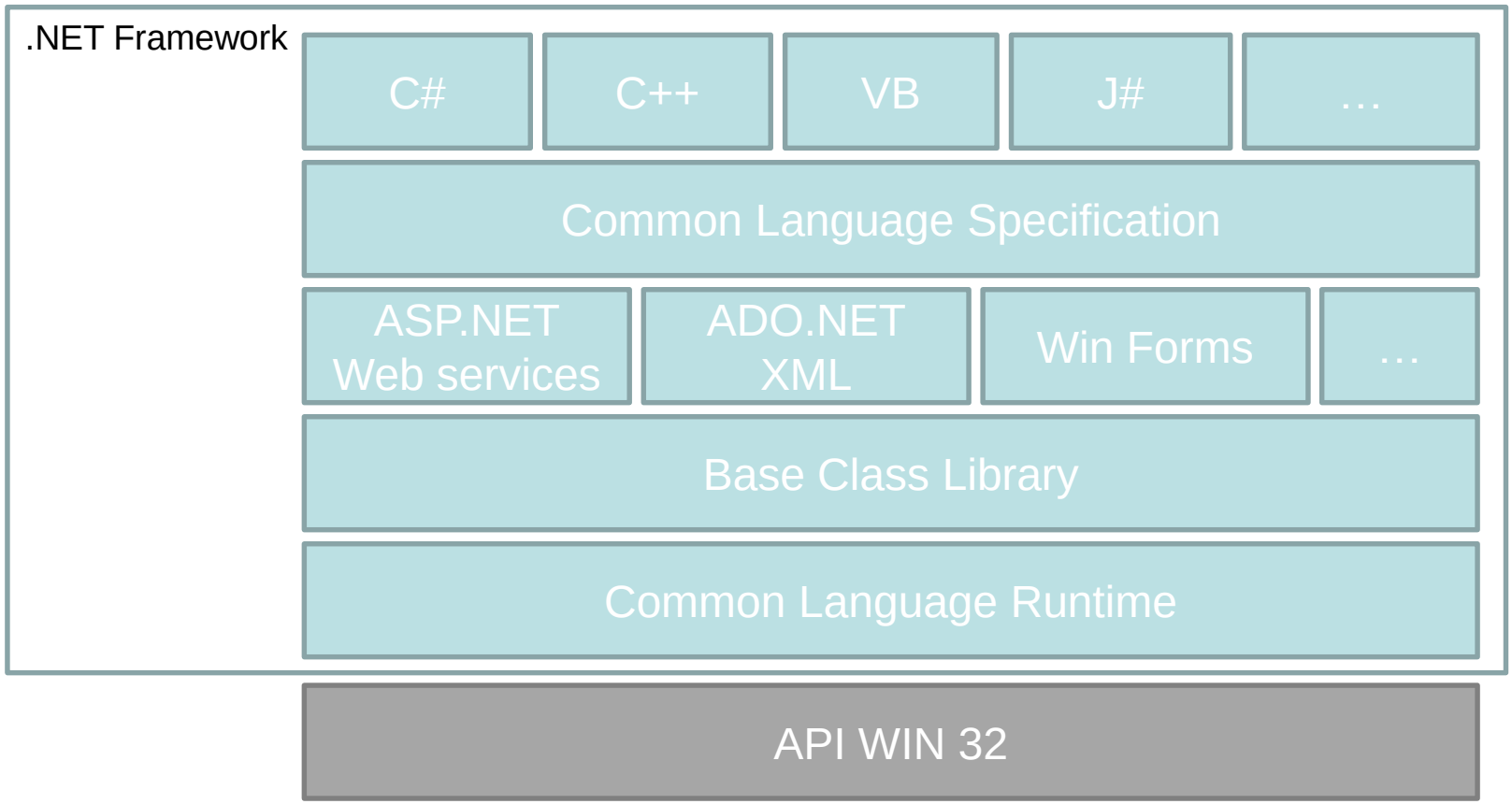
Formation C#

Formation C#

- Framework .NET
- Notion de base du langage
- Programmer Objet en C#
- Les objets standards
- Les expressions Lambda
- Gestion des Exceptions
- Les entrées-sorties
- Mise en place du pattern DAO

Framework .NET

- L'architecture

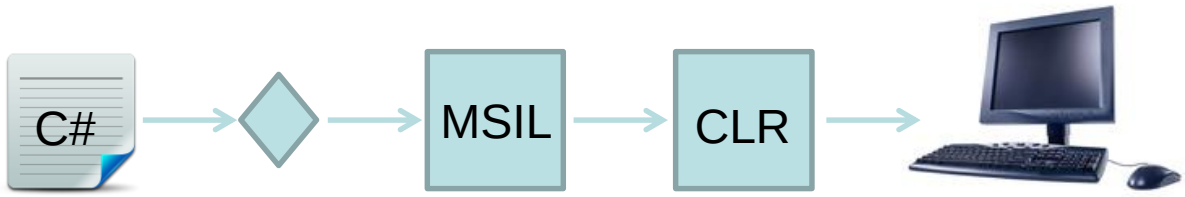


Framework .NET

- Les avantages
  - Facilité d'emploi
  - Classes extensibles
  - Modèles d'applications unifiés
  - Pour les développeurs
    - Développement plus rapide
    - Plus grande fiabilité
    - Basé sur des standards

Framework .NET

- Les composants
  - Le CLR
    - Machine virtuelle de .NET



C# compiler — Intermediate Language — Code natif

  - Les librairies
    - System.IO, System.Net, System.Text, …
    - System.Data, System.Xml
    - …

Formation C#

- Framework .NET
- Notion de base du langage
- Programmer Objet en Java
- Les objets standards
- Les expressions Lambda
- Gestion des Exceptions
- Les entrées-sorties
- Mise en place du pattern DAO

Notion de base du langage

- Orienté objet
  - Basé sur la notion de classe
    - Respecte les principes d'abstraction, d'encapsulation et de polymorphisme
  - Particularités
    - Héritage simple, pas d'héritage multiple (évite les pbs de duplication d'attributs, conflits entre méthode…)
    - Notion d'interface

Notion de base du langage

- 1er Exemple :

```
namespace Hello
{
    public class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Salut à toi !");
            Console.Read();
        }
    }
}
```

  - public class Program
    - Nom de la classe
  - static void main()
    - La fonction principale équivalent à la fonction main du java
  - String[] args
    - Permet de récupérer des arguments transmis au programme au moment de son lancement
  - Console.WriteLine(« Salut … ")
    - Méthode d'affichage dans la fenêtre console

Syntaxe

- Proche du Java
  - Une instruction se termine par ;
  - Le bloc d'instructions : délimité par { et }
  - La condition booléenne : délimitée par (et)
  - Les tableaux : notation []
- Les commentaires

```
/* Voici un commentaire
 sur plusieurs lignes */

//Voici un commentaire sur une seule ligne

///Pour documenter son code
///en utilisant certains tags de documentation
```

Syntaxe

- Les types de données primitifs
  - Occupent une place fixe en mémoire réservée à la déclaration
    - object : type de base de tous les autres types
    - string : séquence de caractère
    - sbyte, byte : entier signé ou non (de -128 à 127, de 0 à 255)
    - int, uint : entier signé ou non (taille : 2^{32})
    - long, ulong : entier signé (taille : 2^{64})
    - float, double: type virgule flottante (taille : 2^{32},2^{64})
    - char : caractère
    - bool : type booléen
  - A la différence de Java, ce sont des objets
    - Il est donc possible d'appeler une méthode sur un type de données primitif

```
int i = 10;
Console.WriteLine(i.ToString());
```

Syntaxe

- Les types « Nullables »
  - Cela apporte la possibilité de travailler avec des types primitifs non initialisés

```
public class Test
{
    public int i;
    public int? j;
}

static void Main(string[] args)
{
    Test t = new Test();
    Console.WriteLine(t.i.ToString()); //retourne 0
    Console.WriteLine(t.j.ToString()); //retourn ""
    if (t.i == null)
    {
        Console.WriteLine("i est null");
        //On ne passe pas ici car i n'est pas null, c'est un int et non un int?
    }
    if (t.j == null)
    {
        Console.WriteLine("j est null");
    }
    //passage d'un int? à un int :
    int k = t.j ?? 0;  //k reçoit la valeur de t.j s'il n'est pas null, sinon reçoit 0;

    Console.Read();
}
```

Syntaxe

- Déclaration et initialisation d'une variable

```
int i; // Déclaration : type nom

i = 10; // Affectation : nom=valeur

int j = 20; //Combinaison : type nom=valeur
```

- Constantes
  - Ce sont des variables dont la valeur ne peut être affectée qu'une fois
  - Elles ne peuvent plus être modifiées
  - Elles sont définies avec le mot clé **const**

```
const int i = 10;

i = 12; // erreur : i est une constante
```

Syntaxe

- Opérateurs arithmétiques
  - Unaires : -, --, ++
  - Binaires : +, -, *, /, %
- Opérateurs logiques
  - !, &, |, &&, ||
- Opérateurs de comparaison
  - <, >, <=, >=, ==, !=

```
int n = 6;
int m = -n; // m = -6
n++; //n=7
int i = n / 4; // i=1;
double j = n / 4; // j=1.0
j = n / 4.0; // j=1.75
string texte = null;
if (texte != null & texte.Length > 4)
{
    //Les 2 tests sont analysés
    // => NullReferenceException
}
if (texte != null && texte.Length > 4)
{
    //Le 2e test est analysé que si le 1er est vrai
    //ceci va s'exécuter correctement
}
```

```
int i = 6; int j = 6;
if (i == j)
{
    //types primitifs => la comparaison est vraie
}
Voiture v1 = new Voiture("123 AXB 31");
Voiture v2 = new Voiture("123 AXB 31");
if (v1 == v2)
{
    //La comparaison est fausse !
}
```

Syntaxe

- Les tableaux

```
//Déclaration
int[] tableau1;
int[,] tableau2; //tableau à 2 dimensions
//Dimensionnement :
tableau1 = new int[3];
//Initialisation :
tableau1[0] = 10; //les indices commences à zéro
tableau1[1] = 20;
tableau1[2] = 30;
//Combinaison :
int[] tableau3 = { 10, 20, 30 };

//Nombre d'élément :
int taille = tableau1.Length;

//Parcours d'un tableau :
for (int i = 0; i < tableau1.Length; i++)
{
    Console.WriteLine(tableau1[i].ToString());
}
```

Syntaxe

- Le type string

```
//Création :
string texte = "Bonjour";
//Longueur d'une string :
int longueur = texte.Length; //longueur=7
//Conversion vers une string
int i = 10;
string dix = i.ToString();
// ou :
dix = "" + i;
//Conversion inverse :
int j = int.Parse(dix);
// ou :
int k;
bool ok = int.TryParse(dix,out k);
//Les comparaisons :
string s1 = "abc";
string s2 = "abc";
if (s1 == s2) Console.WriteLine("s1 = s2");
if (s1.Equals(s2)) Console.WriteLine("s1 = s2");
//Méthode de recherche :
texte.StartsWith("Bon"); //retourne vrai si commence par Bon
texte.EndsWith("r");//retourne vrai si se termine par r
texte.IndexOf("on");//retourne 1 : l'index de la 1ère occurence de on
                    //-1 si pas trouvé
texte.LastIndexOf("o");//retourne 4 : l'index de la dernière occurence de o
//L'édition :
texte.ToUpper(); //retourne BONJOUR
texte.Substring(0, 3); //retourne Bon
texte.Replace("on", "aa"); //retourne Baajour
"Bonjour    ".Trim(); //retourne Bonjour
//Concaténation :
string chaine = texte + " à vous " + dix;
```

Structures de contrôle

- La condition

```
if (condition)
{
    //instructions;
}


if (condition)
{
    //instructions;
}
else
{
    //instructions;
}


if (condition1)
{
    //instructions;
}
else if (condition2)
{
    //instructions;
}
else
{
    //instructions;
}
```

Structures de contrôle

- Le traitement switch-case

```
switch (texte)
{
    case "Bonjour":
        //instructions;
        break;
    case "Salut":
        //instructions;
        break;
    default:
        //instructions;
        break;
}
```

- La boucle for et foreach

```
for (initialisation; condition; action)
{
    //instrutions;
}
//exemple :
for (int i = 0; i < 10; i++)
{
    //instructions;
}

int[] tableau = { 10, 20, 30 };
//foreach :
foreach (int item in tableau)
{
    Console.WriteLine(item);
}
```

Structures de contrôle

- La boucle while et la boucle do-while

```
while (condition)
{
    //instructions;
}

do
{
    //instructions;
} while (condition);
```

- Action sur une boucle
  - **break** permet de sortir de la boucle
  - **continue** permet d'aller directement à l'évaluation suivante

Travaux pratiques – TP1

- Créer le programme Hello avec paramètre
  - Le nom est en paramètre
  - Affiche : *Hello Jean* (Jean passé en paramètre)
- Créer le programme TrieTableau
  - Tableau d'entier : 4, 1, 7, 3, 2, 9
  - Trie et Affiche le tableau trié

Formation C#

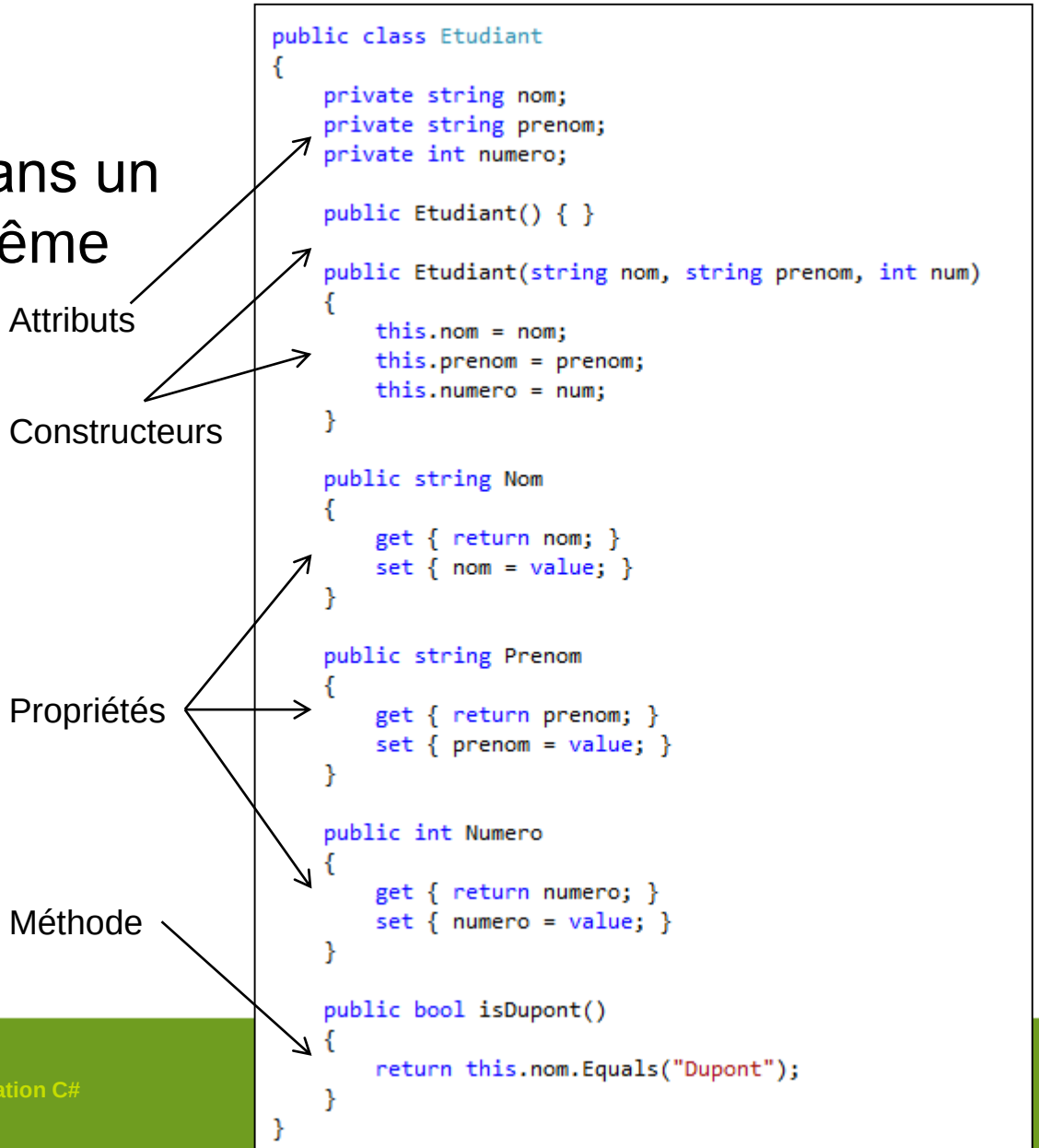
- Framework .NET
- Notion de base du langage
- Programmer Objet en C#
- Les objets standards
- Les expressions Lambda
- Gestion des Exceptions
- Les entrées-sorties
- Mise en place du pattern DAO

Classe

- Définition
  - Une classe est constituée :
    - de données qu'on appelle des attributs
    - de procédures et/ou des fonctions qu'on appelle des méthodes
  - Une classe est un modèle de définition pour des objets
    - Ayant même structure (même ensemble d'attributs)
    - Ayant même comportement (même méthodes)
  - Les objets sont des représentations dynamiques de la classe (instanciation)
    - Une classe permet d'instancier (créer) plusieurs objets
    - Chaque objet est instance d'une classe et une seule

Classe

- Déclaration
  - Le code s'écrit dans un fichier qui porte le même nom (conseillé)




```
public class Etudiant
{
    private string nom;
    private string prenom;
    private int numero;

    public Etudiant() { }

    public Etudiant(string nom, string prenom, int num)
    {
        this.nom = nom;
        this.prenom = prenom;
        this.numero = num;
    }

    public string Nom
    {
        get { return nom; }
        set { nom = value; }
    }

    public string Prenom
    {
        get { return prenom; }
        set { prenom = value; }
    }

    public int Numero
    {
        get { return numero; }
        set { numero = value; }
    }

    public bool isDupont()
    {
        return this.nom.Equals("Dupont");
    }
}
```

Classe

- Les attributs
  - Variables globales dans la classe
- Les constructeurs
  - Méthode d'initialisation qui porte le nom de la classe
  - Il peut y avoir plusieurs constructeurs pour une classe

```
public class Etudiant
{
    private string nom;
    private string prenom;
    private int numero;

    public Etudiant() { }

    public Etudiant(string nom, string prenom, int num)
    {
        this.nom = nom;
        this.prenom = prenom;
        this.numero = num;
    }
```

Classe

- Les méthodes
  - Procédures/fonctions qui permettent de manipuler l'objet
  - Les paramètres sont passés
    - Par valeur pour les types primitifs
    - Par référence pour les objets
    - Possibilité d'utiliser les mots clés **ref** ou **out**

Classe Etudiant

```
public void ajoute(int nombre)
{
    nombre = nombre+1;
    this.numero = nombre;
}

public void ajouteRef(ref int nombre)
{
    nombre = nombre + 1;
    this.numero = nombre;
}

public void ajouteOut(int nombre, out string sonNom)
{
    nombre = nombre + 1;
    this.numero = nombre;
    sonNom = this.Nom;
}
```

```
Etudiant e = new Etudiant("CELAIRE","Jacques",100);

int i = 10;
e.ajoute(i);
Console.WriteLine("i=" + i); //i=10
e.ajouteRef(ref i);
Console.WriteLine("i=" + i); //i=11
string nomEtudiant;
e.ajouteOut(i, out nomEtudiant);
Console.WriteLine("i=" + i + " nomEtudiant:"+nomEtudiant);
//i=10 nomEtudiant:CELAIRE
```

Classe

- Les propriétés
  - Méthodes spéciales qui fournit un mécanisme pour la lecture, l'écriture ou le calcul de la valeur d'un champ privé
  - Elles peuvent être utilisées comme s'il s'agissaient de membres de données publics
  - Ce sont les accesseurs : l'équivalent des getters et setters en Java

```
public string Nom
{
    get { return nom; }
    set { nom = value; }
}

public string Prenom
{
    get { return prenom; }
    set { prenom = value; }
}
```

```
Etudiant e = new Etudiant();

e.Nom = "CELAIRE";
e.Prenom = "Jacques";
Console.WriteLine(e.Prenom + " " + e.Nom);
```

Classe

- Contrôle d'accès
  - Il est possible de préciser l'accès aux classes, attributs, méthodes et constructeurs:
    - *private* : accessible uniquement à l'intérieur de la classe
    - *protected* : accessible par les sous-classe
    - *public* : accessible par n'importe quelle classe
    - <aucun> : accessible par toutes les classes du *namespace* de la classe

Objet

- Un objet est instance d'une seule classe
- Tout objet est manipulé et identifié par sa référence
  - Utilisation de pointeur caché
    - « a = b » signifie a devient identique à b

    Les deux objets a et b sont identiques et toute modification de a entraîne celle de b

    - « a == b » retourne « true » si les deux objets sont identiques

    C'est-à-dire si les références sont les mêmes, cela ne compare pas les attributs

Objet

- Manipulation

```
//Déclaration :
Etudiant e1;
//Création et allocation mémoire :
e1 = new Etudiant();
//Déclaration et création en une seule ligne
Etudiant e2 = new Etudiant("BREILLE", "Jean", 759);
//Utilisation :
e1.Nom = "CELAIRE";
e1.Prenom = "Jacques";
Console.WriteLine(e1.Prenom + " " + e1.Nom);

//Attention à l'affectation :
Etudiant f = e1;
f.Nom = "DUPONT";
Console.WriteLine(e1.Prenom + " " + e1.Nom);
//Affichage sur la console ?
```

Objet

- Un attribut peut être une instance d'une autre classe

```
public class Etudiant
{
    private string nom;
    private string prenom;
    private int numero;
    private Formation formation;

    public Etudiant() { }

    public Etudiant(string nom, string prenom, int num, Formation formation)
    {
        this.nom = nom;
        this.prenom = prenom;
        this.numero = num;
        this.formation = formation;
    }
```

Objet

- Gestion des objets

```
Etudiant e = new Etudiant();
//Récupérer le type d'un objet :
Type t = e.GetType(); // retourne un objet Type

//Tester le type
if (e is Etudiant) { Console.WriteLine("e est un étudiant"); }
if (e.GetType() == typeof(Etudiant)) { Console.WriteLine("e est un étudiant"); }

//Cast
Etudiant e1 = (Etudiant)objetInconnu; //Peut déclencher une Exception

Etudiant e2 = objetInconnu as Etudiant;
//e2 est un Etudiant si le Cast a résussi, sinon il est null
```

Objet

- Variables statiques
  - Ce sont des constantes liées à une classe

déclaration →

```
public class Formation
{
    public static string DUT = "01";
    public static string LP = "02";

    private string code;
    private string nom;

    public Formation() { }

    public Formation(string code)
    {
        this.code = code;
    }
```

utilisation →

```
Formation form = new Formation(Formation.LP);
```

Objet

- Méthodes statiques
  - Ce sont des méthodes qui ne s'intéressent pas à un objet particulier

déclaration →

```
public static Formation getDUT()
{
    return new Formation(DUT);
}
```

utilisation →

```
Formation form = Formation.getDUT();
```

Objet

- Classes statiques
  - Ne peut pas être instanciée
  - Ne peut contenir que des attributs et méthodes statiques

déclaration →

```
public static class Utilitaire
{
    public static float TAUX_TVA_REDUIT = 5.5F;
    public static float TAUX_TVA_NORMAL = 20F;

    public static float AjouteTVA(float prixht, float taux)
    {
        return prixht * (1 + (taux / 100));
    }
}
```

utilisation →

```
float prixProduitHT = 50;
float prixProduitTTC = Utilitaire.AjouteTVA(prixProduitHT,Utilitaire.TAUX_TVA_NORMAL);
Console.WriteLine("Avec une TVA à " + Utilitaire.TAUX_TVA_NORMAL + "% le prix TTC = " + prixProduitTTC);
```

Espace de noms

- Permet le regroupement de classes et d'interfaces logiquement liées
- Les *namespace* sont construits en utilisant une notation pointée
- Permet de résoudre les conflits de nom
  - Exemples :
    - System.text, System.IO …
    - MonProgramme.Form

Espace de noms

- Visibilité
  - Par défaut, une classe n'est accessible que par les classes du même *namespace*
- Utilisation

```
//Utilisation avec le nom complet : namespace + nom de la classe :
Hello.Model.Etudiant etudiant = new Hello.Model.Etudiant();
// => écriture très lourde : préferer l'utilisation du using
```

```
using Hello.Model;

        ...

    Etudiant etudiant = new Etudiant();
```

Les Héritages

- C# ne permet que l'héritage simple
  - Une seule classe parent

```
public class Etudiant : Personne
```

  - Les méthodes de la classe dérivée peuvent accéder aux attributs *public* et *protected* de la classe parent
  - Le mot de clé *base* représente la classe parent

```
public Etudiant(string nom, string prenom, int num, Formation formation)
{
    base.Nom = nom; //utilisation des attributs de Personne
    base.Prenom = prenom;
    this.numero = num;
    this.formation = formation;
}

//Avec l'appel du constructeur de Personne :
public Etudiant(string nom, string prenom, int num, Formation formation) : base(nom,prenom)
{
    this.numero = num;
    this.formation = formation;
}
```

Les Héritages

- Surcharger une méthode
  - Nécessite d'être déclarée virtuelle dans la classe parente. Mot clé : *virtual*

```
public virtual string getInfos()
{
    return this.prenom+" "+this.nom;
}
```

  - Dans la classe fille, mot clé : *override*

```
public override string getInfos()
{
    return this.numero+ " : "+base.getInfos();
}
```

Les Héritages

- Toutes les classes héritent de *object*
  - => Possibilité de surcharger ses méthodes
    - Fortement conseillé !

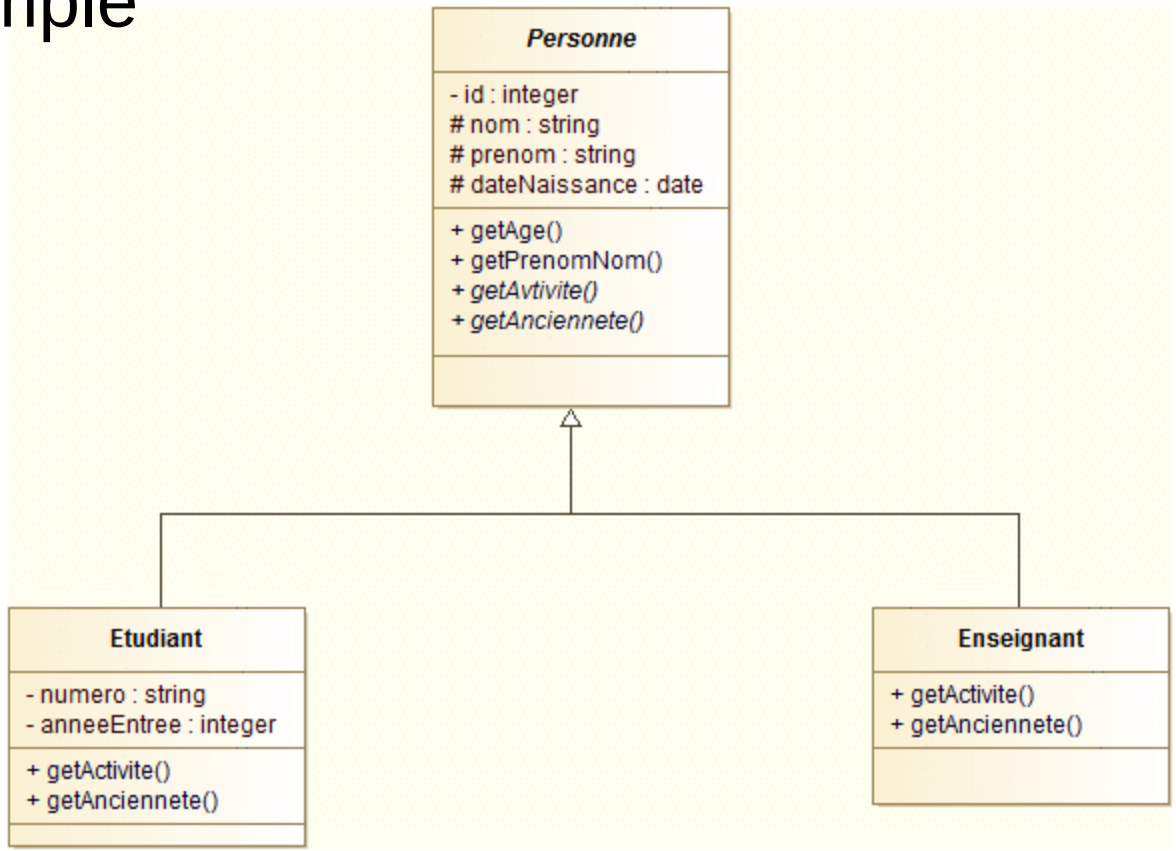
```
public override bool Equals(object obj)
{
    // si le paramètre ne peut pas être casté, on retourne false
    Etudiant e = obj as Etudiant;
    if ((object)e == null) return false;

    return this.numero.Equals(e.numero);
}

public override string ToString()
{
    return this.Prenom+" "+this.Nom;
}

public override int GetHashCode()
{
    return this.numero.GetHashCode();
}
```

Les classes abstraites

- Intérêt
  - Permet de factoriser du code
- 3 règles :
  - Si une seule des méthodes d'une classe est abstraite, alors la classe doit être abstraite
  - On ne peut pas instancier une classe abstraite
  - Toutes les classes filles héritant de la classe mère abstraite doivent implémenter toutes ses méthodes

Les classes abstraites

- Exemple



Les classes abstraites

- Exemple

```
public abstract class Personne
{
    private int id;
    protected string nom;
    protected string prenom;
    protected DateTime dateNaissance;

    public int getAge()
    {
        return DateTime.Now.Year - dateNaissance.Year;
    }

    public string getPrenomNom()
    {
        return this.prenom+" "+this.nom;
    }

    public abstract string getActivite();
    public abstract int getAnciennete();
}
```

```
public class Etudiant : Personne
{
    private int numero;
    private Formation formation;
    private int anneeEntree;

    public override string getActivite()
    {
        return "Etudie";
    }

    public override int getAnciennete()
    {
        return DateTime.Now.Year - anneeEntree;
    }
```

```
public class Enseignant : Personne
{
    private Contrat contrat;

    public override string getActivite()
    {
        return "Enseigne";
    }

    public override int getAnciennete()
    {
        return DateTime.Now.Year - contrat.getDebutContrat();
    }
}
```

Les interfaces

- Sorte de classes abstraites sans aucune méthode implémentée
- Intérêt
  - Etablie un contrat avec la classe qui l'implémente
  - Une classe ne peut pas hériter de plusieurs classes
  - Une classe peut implémenter plusieurs interfaces
  - Une interface peut étendre plusieurs interfaces
- Particularités
  - Une interface ne possède pas d'attribut
  - Les interfaces ne sont pas instanciables (comme les classes abstraites)

Les interfaces

- Mise en œuvre
  - Mot clé *interface*
  - Déclaration des signatures des méthodes uniquement

```
public interface ISalarie
{
    double getSalaire();
}
```

- Utilisation

```
public class Titulaire : Enseignant, ISalarie
```

Les interfaces

- Exemple

```
public interface ISalarie
{
    double getSalaire();
}
```

```
public class Titulaire : Enseignant, ISalarie
{
    private int indice;

    public double getSalaire()
    {
        return this.indice * Constantes.VALEUR_INDICE * 12;
    }
}
```

```
public class Vacataire : Enseignant, ISalarie
{
    private float salaireMensuel;

    public double getSalaire()
    {
        return salaireMensuel * 12;
    }
}
```

```
Enseignant[] enseignants = new Enseignant[3];
enseignants[0] = new Titulaire();
enseignants[1] = new Titulaire();
enseignants[2] = new Vacataire();

foreach (ISalarie item in enseignants)
{
    Console.WriteLine(item.getSalaire());
}
```

La documentation

- Intérêts
  - Rédaction de la documentation au format XML exploitable ensuite par Intellisense dans Visual Studio
- Utilisation
  - Précédé de ///
  - Utilisation de tags prédéfinis permettant de typer certaines informations

La documentation

- Exemple de tags :
  - <summary> : description de l'élément
  - <param> : Permet de documenter les paramètres d'une fonction
  - <return> : Permet de documenter la valeur de retour d'une fonction
  - <Exception> : Permet d'informer sur le(s) type(s) d'exception(s)
  - <exemple> : Permet de donner un exemple d'utilisation

Plus d'info ici :
http://vincentlaine.developpez.com/tuto/dotnet/comdoc/

La documentation

- Exemple

```
/// <summary>
/// Constructeur de la classe Edudiant
/// </summary>
/// <param name="nom">Nom de l'étudiant</param>
/// <param name="prenom">Prénom de l'étudiant</param>
/// <param name="num">Numéro de la carte étudiant</param>
/// <param name="formation">Classe Formation</param>
public Etudiant(string nom, string prenom, int num, Formation formation)
{
    base.nom = nom; //utilisation des attributs de Personne
    base.prenom = prenom;
    this.numero = num;
    this.formation = formation;
}
```

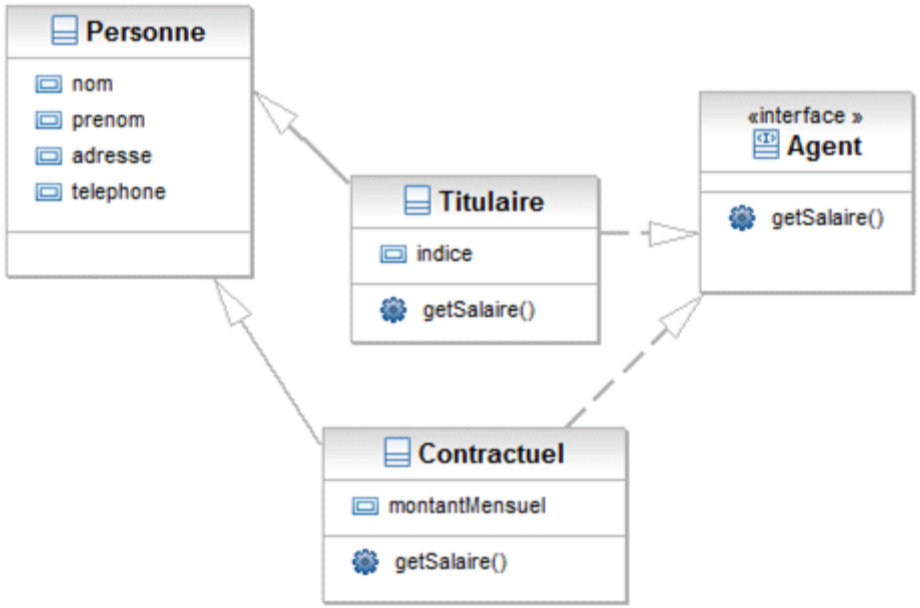
```
Etudiant e = new Etudiant("CELAIRE", "Jacques",
```

▲ 2 sur 2 ▼ Etudiant.Etudiant(string nom, string prenom, **int num**, Formation formation)

Constructeur de la classe Edudiant

num: *Numéro de la carte étudiant*

Travaux pratiques – TP2



Travaux pratiques – TP2

- Création de la classe Titulaire qui hérite de Personne
  - Créer les attributs et les méthodes
  - Dans la méthode main() de Program :
    - Création d'un titulaire
    - Lui affecter un indice
    - Affiche sur la console son nom et son salaire

      Valeur du point d'indice : 55,5635 € brut annuel

Travaux pratiques – TP2

- Création de la classe Contractuel
  - Créer l'interface IAgent avec la méthode getSalaire()
  - Faire implémenter cette interface sur les classes Titulaire et Contractuel
  - Implémenter la méthode getSalaire() sur ces 2 classes
  - Calculer la masse salariale annuelle dans la méthode main() :
    - Création d'un tableau d'IAgent[]
    - Y ajouter des titulaires et des contractuels
    - En parcourant le tableau :
      - Augmenter d'un point l'indice les titulaires
      - Afficher sur la console la masse salariale annuelle

Formation C#

- Framework .NET
- Notion de base du langage
- Programmer Objet en C#
- Les objets standards
- Les expressions Lambda
- Gestion des Exceptions
- Les entrées-sorties
- Mise en place du pattern DAO

StringBuilder (system.Text)

- Version modifiable d'une chaine de caractères

```
StringBuilder sb = new StringBuilder();
//Méthode Append
sb.Append("Salut");
sb.Append(" à vous ");
string s1 = "Jean";
string s2 = "Jacques";
//Méthode AppendFormat
sb.AppendFormat("mes amis {0} et {1}",s1,s2);
//Méthode ToString :
Console.WriteLine(sb.ToString());
//Méthode Remove
sb.Remove(5, 7); //Remove(startIndex, length)
//Méthode Replace
sb.Replace("Salut", "Bonjour");
Console.WriteLine(sb.ToString());
```

Les expressions régulières

- Utilisation de la classe Regex
  - Using System.Text.RegularExpression
  - Permet de vérifier la syntaxe d'une chaine

Symbole	Correspondance	Exemple
\	Caractère d'échappement	[\.] contient un "."
^	Début de ligne	^b$ contient uniquement b
.	N'importe quel caractère	^.$ contient un seul caractère
$	Fin de ligne	er$ finit par "er"
|	Alternative	^(a|A) commence par a ou A
( )	Groupement	^((a)|(er)) commence par a ou er
-	Intervalle de caractères	^[a-d] commence par a,b,c ou d
[ ]	Ensemble de caractères	[0-9] contient un chiffre
[^]	Tout sauf un ensemble de caractères	^[^a] ne commence pas par a
+	1 fois ou plus	^(a)+ commence par un ou plusieurs a
?	0 ou 1 fois	^(a)? commence ou non par un a
*	0 fois ou plus	^(a)* peut ou non commencer par a
{x}	x fois exactement	a{2} deux fois "a"
{x,}	x fois au moins	a{2,} deux fois "a" au moins
{x, y}	x fois minimum, y maximum	a{2,4} deux, trois ou quatre fois "a"

Les expressions régulières

- Utilisation de la classe Regex

Alias	Correspondance	Equivalence
\n	Caractère de nouvelle ligne	
\r	Caractère de retour à la ligne	
\t	Caractère de tabulation	
\s	Caractère d'espacement (espace, tabulation, saut de page, etc)	[\f\n\r\t\v]
\S	Tout ce qui n'est pas un espacement	[^\f\n\r\t\v]
\d	Un chiffre	[0-9]
\D	Tout sauf un chiffre	[^0-9]
\w	Un caractère alphanumérique	[a-zA-Z0-9_]
\W	Tout sauf un caractère alphanumérique	[^a-zA-Z0-9_]
\n	Caractère en octal ex:\001 ==> " 1 "	
\xn	Caractère en hexadécimal ex: \x41 ==>" A "	

Les expressions régulières

- Utilisation de la classe Regex
  - Exemple :

```
//Déclaration de l'expression régulières
string formatDate = @"^(([1-2]\d|0[1-9]|[3][0-1])\/([0][1-9]|[1][0-2])\/[1-2][0-9]\d{2})$";
Regex rgx = new Regex(formatDate);
//Vérification de la variable
if (rgx.IsMatch(chaineAVerifier))
{
    Console.WriteLine("La chaine saisie est une date au format dd/mm/yyyy");
}
```

http://lgmorand.developpez.com/dotnet/regex/

Les collections

- Plusieurs Objets pour grouper un ensemble d'éléments

  Elles sont toutes dans System.Collections

  - List<T>
    - Séquence d'éléments ordonnée par indice
    - Peut contenir des éléments en double
  - HashSet<T>
    - Ensemble d'éléments uniques (pas de doublons)
    - Fonctionne si GetHashCode est surchargée
  - Dictionary<TKey,TValue>
    - Stock des paires clé-valeur
    - Pas de doublons pour les clés

Les collections

- Ce sont des classes qui implémentent toutes l'interface Icollection<T>
  - Méthodes et propriétés communes
    - Add(), Count, Contains(), Remove(), …

```
List<Etudiant> listEtudiants = new List<Etudiant>();
HashSet<Etudiant> setEtudiants = new HashSet<Etudiant>();
Dictionary<int, Etudiant> dico = new Dictionary<int, Etudiant>();

Etudiant e1 = new Etudiant(); e1.Numero = 1;
Etudiant e2 = new Etudiant(); e2.Numero = 2;
Etudiant e3 = new Etudiant(); e3.Numero = 1;

listEtudiants.Add(e1);
listEtudiants.Add(e2);
listEtudiants.Add(e3);
Console.WriteLine("nb élément dans la liste : "+listEtudiants.Count); //Résulat ?

setEtudiants.Add(e1);
setEtudiants.Add(e2);
setEtudiants.Add(e3);
Console.WriteLine("nb élément dans la liste : " + setEtudiants.Count); //Résulat ?

dico.Add(0, e1);
dico.Add(1, e2);
dico.Add(5, e3);
Console.WriteLine("nb élément dans la liste : " + dico.Count); //Résulat ?

if (setEtudiants.Contains(e3)) Console.WriteLine("e3 est présent dans setEtudiants");
Console.WriteLine("Numéro de e3 : "+dico[5].Numero);
```

DateTime

- A la différence de java, l'objet DateTime contient toutes les méthodes
  - Pour le stockage :

```
//Création d'une nouvelle date :
DateTime dt1 = new DateTime();
//Création et initialisation de la date :
DateTime dt2 = DateTime.Parse("06/02/1973");
CultureInfo provider = CultureInfo.CurrentCulture;
string format = "dd/MM/yyyy";
DateTime dt22 = DateTime.ParseExact("06/02/1973",format,provider);
DateTime dt3 = DateTime.Now;
DateTime dt4 = DateTime.Today;
```

  - Pour la manipulation :

```
//Ajout de jour, mois, année :
DateTime dt5 = dt2.AddDays(5);
DateTime dt6 = dt2.AddMonths(3);
DateTime dt7 = dt3.AddYears(-15);
//Récupération d'information
DayOfWeek dow =  dt2.DayOfWeek;
int jour = dt2.DayOfYear;
int heure = dt3.Hour;
```

  - Pour l'affichage :

```
//Affichage :
Console.WriteLine(dt1.ToString());
Console.WriteLine(dt2.ToShortDateString());
Console.WriteLine(dt3.ToLongDateString());
Console.WriteLine(dt7.ToShortTimeString());
```

Travaux pratiques – TP3

- Utilisation des collections pour stocker les étudiants
  - Créer la classe Universite
  - Ajouter l'attribut List<Etudiant> etudiants
  - Implémenter une méthode ajoute qui permet d'ajouter un étudiant à la liste
  - Implémenter une méthode getEtudiant qui permet de retrouver un étudiant dans la liste à partir de son numéro
  - Dans une classe Program
    - Créer une université avec 10 étudiants
    - Ajouter un 11e étudiant en utilisant la méthode ajoute
    - Le rechercher avec la méthode getEtudiant

Travaux pratiques – TP3

- Ajouter à la classe Etudiant l'attribut dateDeNaissance
  - Prendre en compte ce nouvel attribut dans le constructeur de la classe
  - Implémenter une méthode affiche() sur la Classe Etudiant afin d'afficher : numero : dd/mm/yyyy
  - Dans la Classe Program
    - Afficher la liste des étudiants de l'université en utilisant la méthode affiche de la classe Etudiant

Formation C#

- Framework .NET
- Notion de base du langage
- Programmer Objet en C#
- Les objets standards
- Les expressions Lambda
- Gestion des Exceptions
- Les entrées-sorties
- Mise en place du pattern DAO

Les expressions Lambda

- Fonction anonyme qui peut contenir des expressions et des instructions
  - Opérateur lambda =>
  - Une expression est toujours constituée de deux parties :
    - Le côté gauche donne les paramètres d'entrées (s'il y en a)
    - Le côté droit donne les instructions de la méthode anonyme

```
Func<int, int> myDelegate = x => x * x;
int square = myDelegate(5); // square vaut 25

Console.WriteLine("5 x 5 = " + square);
```

Les expressions Lambda

- Très pratique pour la gestion des listes
  - Peut remplacer les boucles et les conditions
    - Information sur la liste

```
//Retourne vrai si tous les éléments de la liste vérifient la condition
Boolean tousSupp1000 = personnes.All(p => p.Salaire >= 1000);
//Retourne vrai si au mmoins un élément de la liste vérifie la condition
Boolean aumoinsUn2000 = personnes.Any(p => p.Salaire >= 2000);
//Retourne le nombre d'élément qui vérifie la condition
int nbSupp1500 = personnes.Count(p => p.Salaire >= 1500);
```

    - Action sur les éléments de la liste

```
//Supprime tous les éléments qui vérifient la condition
int nbPersonnesSupprimee = personnes.RemoveAll(p => p.Prenom.Equals("Jean"));
//Exécute l'action spécifiée pour chaque élément de la liste
personnes.ForEach(p => Console.WriteLine(p.ToString()));
personnes.ForEach(p => p.Salaire = p.Salaire + 100);
//Trie les éléments de la liste selon un critère
List<Personne> personnesTriees = personnes.OrderBy(p => p.Salaire).ToList();
```

Les expressions Lambda

- Très pratique pour la gestion des listes
  - Peut remplacer les boucles et les conditions
    - Sélection

```
//Retourne une sous-liste contenant les éléments qui vérifient la condition
List<Personne> toulousains = personnes.Where(p => p.VilleNaiss.Equals("Toulouse")).ToList();
//Regroupe les élément en fonction du critère
var groupe = personnes.GroupBy(p => p.VilleNaiss).ToList();
//Recrée une liste composée d'un nouvel objet
var lesGens = personnes.Select(p => new {nom = p.Prenom+" "+p.Nom, salaireAnnuel = p.Salaire*12}).ToList();

//Tout cela peut se combiner
//Exemple : afficher la moyenne des salaires par ville
var result = personnes.GroupBy(p => p.VilleNaiss).Select(g => new { ville = g.Key, moyenne = g.Average(e => e.Salaire) }).ToList();
result.ForEach(r => System.Console.WriteLine(r.ville+" : "+r.moyenne));
```

Travaux pratiques – TP4

- Exercices
  - Créer une liste d'étudiants : List<Etudiant>
  - Sans jamais utiliser de boucle :
    - Trier et afficher la liste du plus jeune au plus vieux
    - Calculer l'âge moyen
    - Afficher le nombre d'étudiant par année de naissance

Formation C#

- Framework .NET
- Notion de base du langage
- Programmer Objet en C#
- Les objets standards
- Les expressions Lambda
- Gestion des Exceptions
- Les entrées-sorties
- Mise en place du pattern DAO

Les Exceptions

- Définition
  - Une exception est un signal indiquant que quelque chose d'exceptionnelle (comme une erreur) s'est produit
  - Elle interrompt le flot d'exécution normal du programme
- Cela facilite la gestion des erreurs
  - Permet de séparer clairement le code d'exécution normale du code de gestion des erreurs

Les Exceptions

- Bloc try
  - Instructions susceptibles de lever une ou plusieurs exceptions
- Bloc catch
  - Instructions exécutés à la levée de l'exception
  - Plusieurs blocs *catch* peuvent suivre un bloc *try*
- Bloc finally
  - Instructions exécutés en sortie du bloc *try* avec ou sans levés d'exception
  - Si exception levée, les instructions du bloc *finally* seront exécutés après les instructions du bloc *catch*

Les Exceptions

```
try
{
    //Instructions
}
catch (Exception ex)
{
    Console.WriteLine("Erreur dans l'instruction : "+ex.Message);
}
finally
{
    //Insctructions
}
```

Les Exceptions

- Créer son Exception

```
public class EtudiantExisteDejaException : Exception
{
    private Etudiant etudiant;

    public EtudiantExisteDejaException(Etudiant e)
    {
        this.etudiant = e;
    }

    public override string Message
    {
        get
        {
            return "L'étudiant " + etudiant.ToString() + " existe déjà !";
        }
    }
}
```

Les Exceptions

- Instruction *throw*
  - Permet de créer et d'envoyer une exception

```
/// <summary>
/// Ajoute un étudiant dans la liste
/// </summary>
/// <param name="etudiant">de type Etudiant</param>
/// <exception cref="EtudiantExisteDejaException">Si l'étudiant existe déjà dans la liste</exception>
public void ajoute(Etudiant etudiant)
{
    if (etudiants == null) etudiants = new List<Etudiant>();
    if (this.getEtudiant(etudiant.Numero) != null)
    {
        throw new EtudiantExisteDejaException(etudiant);
    }
    else
    {
        etudiants.Add(etudiant);
    }
}
```

Travaux pratiques – TP5

- Créer une Exception pour un étudiant non trouvé dans l'université
  - Si l'étudiant n'est pas trouvé lever cette exception
  - Dans la classe Program chercher un étudiant absent
  - Vérifier que l'exception est bien levée

Formation C#

- Framework .NET
- Notion de base du langage
- Programmer Objet en C#
- Les objets standards
- Les expressions Lambda
- Gestion des Exceptions
- Les entrées-sorties
- Mise en place du pattern DAO

Les entrées/sorties

- Plusieurs classes de System.IO permettent de gérer les flux de données en entrée et en sortie
  - StreamWriter
    - Permet d'écrire dans un fichier texte
  - StreamReader
    - Permet de lire le contenu d'un fichier texte

Les entrées/sorties

```
StreamWriter fluxInfos = null; // le fichier texte
try
{
    fluxInfos = new StreamWriter(@"c:\temp\test.txt");
    // écriture d'une ligne dans le fichier texte
    fluxInfos.WriteLine("C'est facile le C# !");
    fluxInfos.WriteLine("2e ligne");
}
catch (Exception e)
{
    Console.WriteLine("L'erreur suivante s'est produite : " + e.Message);
}
finally
{
    fluxInfos.Close();
}

StreamReader fluxLecture = null; string ligne="";
try
{
    // Mode de lecture ligne par ligne
    fluxLecture = new StreamReader(@"c:\temp\test.txt");
    ligne = fluxLecture.ReadLine();
    while (ligne != null)
    {
        Console.WriteLine(ligne);
        ligne = fluxLecture.ReadLine();
    }
    // Mode de lecture jusqu'à la fin du fichier
    Console.WriteLine("2e lecture");
    fluxLecture = new StreamReader(@"c:\temp\test.txt");
    Console.WriteLine(fluxLecture.ReadToEnd());
}
catch (Exception ex)
{
    Console.WriteLine("L'erreur suivante s'est produite : " + ex.Message);
}
```

Les entrées/sorties

- Manipulation des fichiers et répertoires
  - *File*, *Directory* et *Path* : trois classes statiques

```
string rep = @"c:\formationCSharp";
if (!Directory.Exists(rep))
{
    Directory.CreateDirectory(rep);
}
Console.WriteLine("Date de création du répertoire : "+Directory.GetCreationTime(rep));

string[] fichiers = Directory.GetFiles(@"c:\temp");
StreamReader srFichierText = null;
try
{
    foreach (string item in fichiers)
    {
        if (Path.GetExtension(item).Equals(".txt"))
        {
            srFichierText = File.OpenText(item);
            Console.WriteLine("1ère ligne de " + item + " : " + srFichierText.ReadLine());
            srFichierText.Close();
            File.Move(item, rep+@"\"+Path.GetFileName(item));
        }
    }
}
catch (Exception e)
{
    Console.WriteLine("L'erreur suivante s'est produite : " + e.Message);
}
```

Les entrées/sorties

- Manipulation des fichiers et répertoires
  - En utilisation *FileInfo*, *DirectoryInfo*

```
DirectoryInfo di = new DirectoryInfo(@"C:\formationCSharp2");
if (!di.Exists)
{
    di.Create();
}
Console.WriteLine("Date de création du répertoire :" + di.CreationTime);

FileInfo[] fis = di.GetFiles();
foreach (FileInfo item in fis)
{
    if (item.Extension.Equals(".txt"))
    {
        Console.WriteLine(item.FullName);
    }
    else
    {
        Console.WriteLine(item.Name+" : "+item.LastWriteTime);
    }
}
```

Les entrées/sorties

- Manipulation des fichiers XML
  - System.Xml.XmlDocument
    - Lecture :

```
//Initialisation
XmlDocument xmlDocument = new XmlDocument();
try
{
    //Chargement du fichier XML
    xmlDocument.Load(@"c:\temp\config.xml");
    //récupère le noeud principal
    XmlElement racine = xmlDocument.DocumentElement;
    //récupère le 1er noeud du noeud principal
    XmlNode modules = racine.FirstChild;
    XmlNode module3 = modules.ChildNodes[2]; // récupère le 3e fils du noeud modules
    //Récupère la valeur d'un attribut
    string name = modules.Attributes["name"].Value;
    //Récupère le texte contenu dans le noeud module (<module name="param">5</module>) :
    string id = modules.InnerText;
}
catch (Exception e)
{
    Console.WriteLine("Problème lors de la lecture du fichier xml :" + e.Message);
}
```

Les entrées/sorties

- Manipulation des fichiers XML
  - System.Xml.XmlDocument
    - Ecriture :

```
//Initialisation
XmlDocument fichierXML2 = new XmlDocument();
//Chargement du fichier xml
try
{
    //Ajout de la Ligne de déclaration :
    XmlDeclaration xmlDeclaration = fichierXML2.CreateXmlDeclaration("1.0", "UTF-8", null);
    XmlElement root = fichierXML2.DocumentElement;
    fichierXML2.InsertBefore(xmlDeclaration, root);
    //Création du premier élement :
    XmlElement programme = fichierXML2.CreateElement("programme");
    //ajout au début du fichier :
    fichierXML2.AppendChild(programme);
    //Création du 2e élément :
    XmlElement options = fichierXML2.CreateElement("options");
    //Ajout du 2e élément dans les fils de l'élément programme :
    programme.AppendChild(options);
    //Création d'un 3e élément :
    XmlElement option = fichierXML2.CreateElement("option");
    //Définition des attributs :
    option.SetAttribute("taille", "50");
    option.SetAttribute("position", "10,10");
    options.AppendChild(option);
    //Sauvegarde :
    fichierXML2.Save("C:\\temp\\test.xml");
}
catch (Exception e)
{
    Console.WriteLine("Problème lié au fichier XML :" + e.Message);
}
```

Les entrées/sorties

- Sérialisation d'objet en fichier XML
  - System.Xml.XmlSerializer

```
Etudiant etudiant = new Etudiant(1, "CELAIRE", "Jacques", 1234, new Formation(Formation.LP));
//Déclaration du XmlSerializer sur du type Etudiant
XmlSerializer xs = new XmlSerializer(typeof(Etudiant));
StreamWriter wr = new StreamWriter(@"C:\formationCSharp\etudiant_" + etudiant.Numero + ".xml");
//Sérialisation de l'obet etudiant
xs.Serialize(wr, etudiant);

Enseignant enseignant = new Enseignant(2, "BREILLE", "Jean", new Contrat("05"));

List<Personne> personnes = new List<Personne>();
personnes.Add(etudiant);
personnes.Add(enseignant);

//Déclaration du XmlSerializer sur du type List<Personne>
XmlSerializer xs2 = new XmlSerializer(typeof(List<Personne>));
StreamWriter wr2 = new StreamWriter(@"C:\formationCSharp\personnes.xml");
//Sérialisation de la liste de personne
xs2.Serialize(wr2, personnes);
```

Pour prendre en compte les particularités des classes filles :

```
[XmlInclude(typeof(Etudiant))]
[XmlInclude(typeof(Enseignant))]
public abstract class Personne
{
```

Les entrées/sorties

- Sérialisation d'objet en fichier XML
  - Résultat du code précédent :

```xml
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfPersonne xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Personne xsi:type="Etudiant">
    <Id>1</Id>
    <Nom>CELAIRE</Nom>
    <Prenom>Jacques</Prenom>
    <DateNaissance>0001-01-01T00:00:00</DateNaissance>
    <Numero>1234</Numero>
    <Formation>
      <Code>02</Code>
    </Formation>
    <AnneeEntree>0</AnneeEntree>
  </Personne>
  <Personne xsi:type="Enseignant">
    <Id>2</Id>
    <Nom>BREILLE</Nom>
    <Prenom>Jean</Prenom>
    <DateNaissance>0001-01-01T00:00:00</DateNaissance>
    <Contrat>
      <Code>05</Code>
    </Contrat>
  </Personne>
</ArrayOfPersonne>
```

Les entrées/sorties

- Désérialisation :

```
//Déclaration du XmlSerializer sur du type List<Personne>
XmlSerializer xsPersonnes = new XmlSerializer(typeof(List<Personne>));
StreamReader srPersonnes = new StreamReader(@"C:\formationCSharp\personnes.xml");
//Désérialisation vers une List<Personne>
List<Personne> listePersonnes = xsPersonnes.Deserialize(srPersonnes) as List<Personne>;
```

Travaux pratiques – TP6

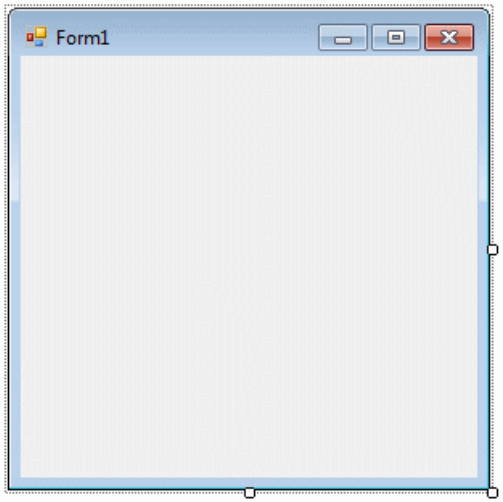
- Ecriture dans un fichier texte
  - Créer une liste de 10 étudiants
  - Ecrire dans un fichier texte, une ligne par étudiant avec : nom;prenom;numero
- Sérialisation d'une liste
  - Sérialiser la liste d'étudiants en XML
- Lecture du fichier XML
  - En utilisant XmlDocument, lire le fichier XML généré par sérialisation pour afficher sur la console le nom et prénom des étudiants

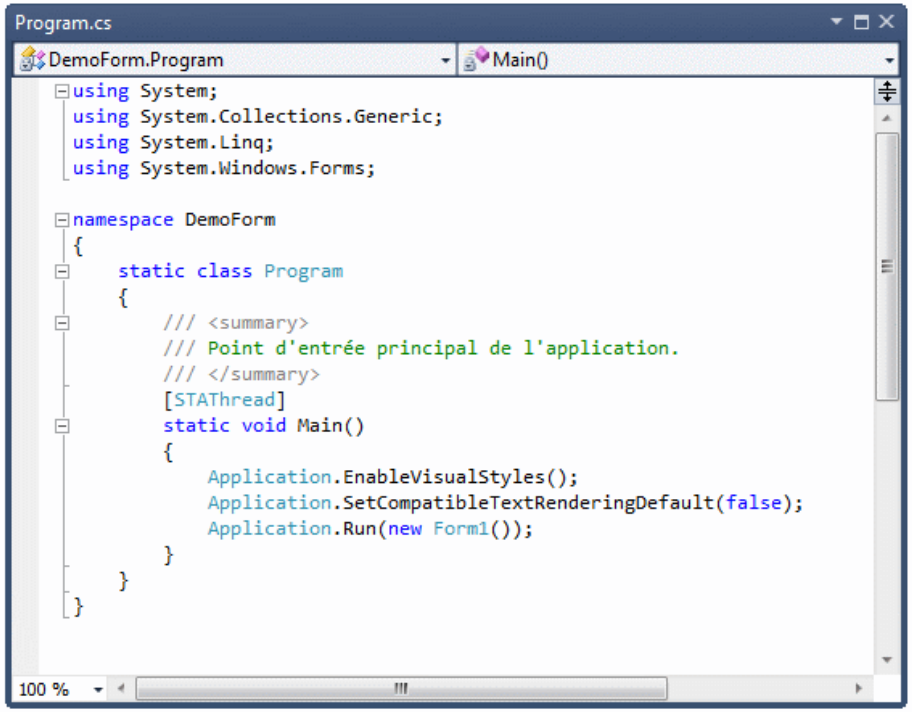
Formation C#

- Framework .NET
- Notion de base du langage
- Programmer Objet en C#
- Les objets standards
- Les expressions Lambda
- Gestion des Exceptions
- Les entrées-sorties
- Mise en place du pattern DAO

IHM avec les Windows Form

- Création d'un projet sous Visual Studio
  - Choisir *Application Windows Forms*





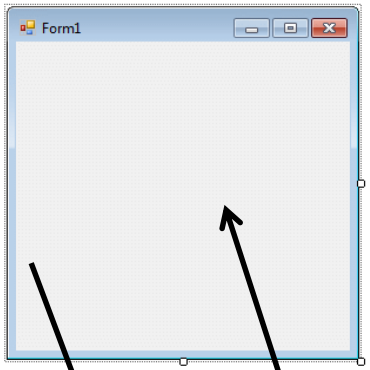
- Une première fenêtre est crée
- La méthode Main() est initialisée pour lancer cette fenêtre
- Le programme est prêt à démarrer

IHM avec les Windows Form

- Une Windows Form = 3 objets
  - L'objet visuel : nomForm.cs [Design]
  - L'objet Designer : nomForm.Designer.cs
    - Classe contenant le code de la partie graphique de la fenêtre :
      - Position des composants
      - Déclaration des évènements
      - …
  - La classe principale : nomForm.cs
    - Classe contenant le code de la partie *intelligente* de la fenêtre

IHM avec les Windows Form

- Une Windows Form = 3 objets



F7

MAJ + F7

Form1.cs | Form1.Designer.cs | Program.cs | Form

DemoForm.Form1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace DemoForm
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

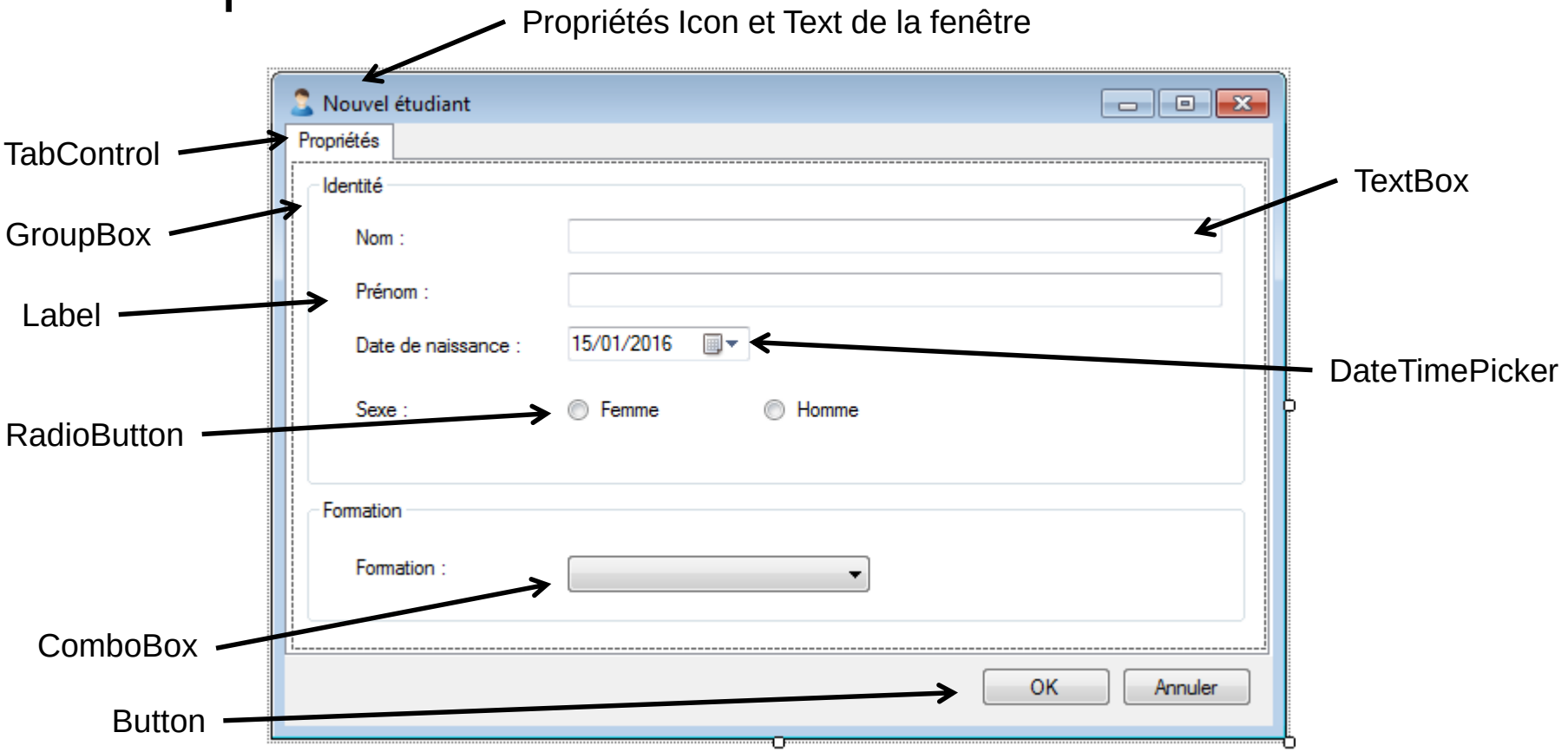
Form1.Designer.cs | Program.cs | Form1.cs [Design]

orm.Form1 | InitializeComponent()

```
espace DemoForm

partial class Form1
{
    /// <summary>
    /// Variable nécessaire au concepteur.
    /// </summary>
    private System.ComponentModel.IContainer components = null;

    /// <summary>
    /// Nettoyage des ressources utilisées.
    /// </summary>
    /// <param name="disposing">true si les ressources managées doivent être supprimées ; sinon, false.</param>
    protected override void Dispose(bool disposing)
    {
        if (disposing && (components != null))
        {
            components.Dispose();
        }
        base.Dispose(disposing);
    }

    #region Code généré par le Concepteur Windows Form

    /// <summary>
    /// Méthode requise pour la prise en charge du concepteur - ne modifiez pas
    /// le contenu de cette méthode avec l'éditeur de code.
    /// </summary>
    private void InitializeComponent()
    {
        this.components = new System.ComponentModel.Container();
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.Text = "Form1";
    }

    #endregion
}
```

IHM avec les Windows Form

- Utiliser la boite à outils pour insérer les composants



IHM avec les Windows Form

- Initialisation des composants

```
public partial class EtudiantForm : Form
{
    private Etudiant etudiant;

    public EtudiantForm()
    {
        InitializeComponent();
    }

    public EtudiantForm(Etudiant etudiant) : this()
    {
        this.Etudiant = etudiant;
        initialisation();
    }

    public Etudiant Etudiant
    {
      get { return etudiant; }
      set { etudiant = value; }
    }
```

…

…

```
public void initialisation()
{
    //Initialisation des TextBox
    textBoxNom.Text = etudiant.Nom;
    textBoxPrenom.Text = etudiant.Prenom;
    //Initialisation du DateTimePicker
    dateTimePickerDateNaissance.Value = etudiant.DateNaissance;
    //Initialisation des RadioButtons
    radioButtonHomme.Checked = (etudiant.Sexe == "M");
    radioButtonFemme.Checked = (etudiant.Sexe == "F");
    //Initialisation de la ComboBox
    comboBoxFormation.DataSource = getFormations();
    comboBoxFormation.SelectedItem = etudiant.Formation;
    comboBoxFormation.DisplayMember = "Nom";
}

public List<Formation> getFormations()
{
    List<Formation> formations = new List<Formation>();
    formations.Add(new Formation("LP", "Licence Pro"));
    formations.Add(new Formation("IUT1", "IUT 1ère année"));
    formations.Add(new Formation("IUT2", "IUT 2e année"));
    return formations;
}
```

Penser à surcharger
la méthode Equals() dans Formation

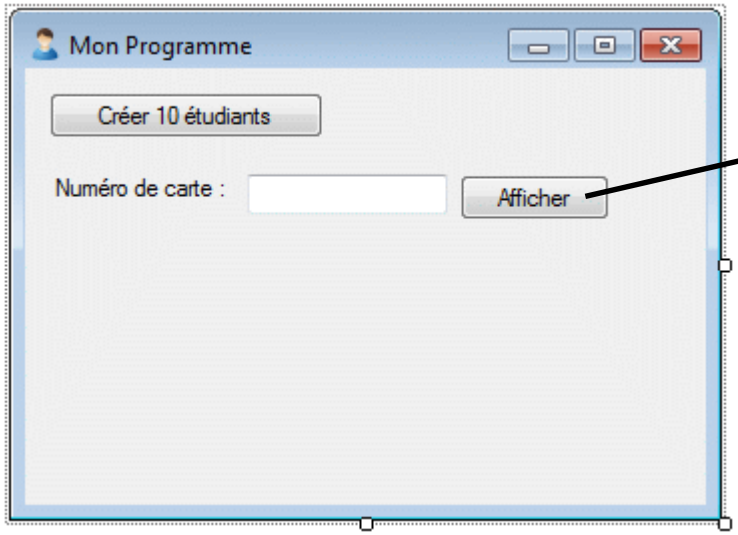
IHM avec les Windows Form

- Codage des boutons et sauvegarde de l'objet

```
private void btnAnnuler_Click(object sender, EventArgs e)
{
    this.Close();
}

private void btnOK_Click(object sender, EventArgs e)
{
    //Message de confirmation
    DialogResult dr = MessageBox.Show("Souhaitez-vous sauvegarder ?", "Sauvegarde", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
    if (dr.Equals(DialogResult.Yes))
    {
        saveEtudiant();
    }
    this.Close();
}

private void saveEtudiant()
{
    //Récupération de la saisie sur un TextBox
    etudiant.Nom = textBoxNom.Text;
    etudiant.Prenom = textBoxPrenom.Text;
    //Récupération de la saisie sur un DateTimePicker
    etudiant.DateNaissance = dateTimePickerDateNaissance.Value;
    //Récupération de la saisie sur un Radiobutton
    etudiant.Sexe = "M";
    if (radioButtonFemme.Checked) etudiant.Sexe = "F";
    //Récupération de la saisie sur un ComboBox
    etudiant.Formation = (Formation)comboBoxFormation.SelectedItem;
    //Message d'information
    MessageBox.Show("Sauvegarde réussie", "Sauvegarde", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

IHM avec les Windows Form

- Appel et ouverture d'une fenêtre




```
private void btnAfficher_Click(object sender, EventArgs e)
{
    int numero = int.Parse(textBoxNumero.Text);
    Etudiant etudiant = getEtudiantByNumero(numero);
    EtudiantForm ef = new EtudiantForm(etudiant);
    ef.ShowDialog();
}

private Etudiant getEtudiantByNumero(int num)
{
    Etudiant etudiant = null;
    foreach (Etudiant item in etudiants)
    {
        if (item.Numero.Equals(num))
        {
            etudiant = item;
            break;
        }
    }
    return etudiant;
}
```

IHM avec les Windows Form

- Utilisation d'un tableau de données :
  - DataGridView depuis la boite à outils

```
dataGridViewEtudiants.DataSource = etudiants;
```

	Numero	Formation	AnneeEntree	Id	Nom	Pre
▶	11	DemoForm.Forma...	0	1	DUPONT_1	Jear
	12	DemoForm.Forma...	0	2	DUPONT_2	Jear
	13	DemoForm.Forma...	0	3	DUPONT_3	Jear
	14	DemoForm.Forma...	0	4	DUPONT_4	Jear
	15	DemoForm.Forma...	0	5	DUPONT_5	Jear
	16	DemoForm.Forma...	0	6	DUPONT_6	Jear
	17	DemoForm.Forma...	0	7	DUPONT_7	Jear
	18	DemoForm.Forma...	0	8	DUPONT_8	Jear

IHM avec les Windows Form

- Utilisation d'un tableau de données :

```
public void afficherTableau()
{
    //Ne pas générer les colonnes automatiquement :
    dataGridViewEtudiants.AutoGenerateColumns = false;
    dataGridViewEtudiants.DataSource = etudiants;
    //Initialisation d'une colonne de type Texte
    DataGridViewTextBoxColumn c1 = new DataGridViewTextBoxColumn();
    c1.Name = "Nom";
    c1.DataPropertyName = "Nom";
    c1.AutoSizeMode = DataGridViewAutoSizeColumnMode.AllCells;
    c1.ReadOnly = true;
    //Initialisation d'une colonne de type Texte
    DataGridViewTextBoxColumn c2 = new DataGridViewTextBoxColumn();
    c2.Name = "Prénom";
    c2.DataPropertyName = "Prenom";
    c2.AutoSizeMode = DataGridViewAutoSizeColumnMode.AllCells;
    //Initialisation d'une colonne de type Combo
    DataGridViewComboBoxColumn  c3 = new DataGridViewComboBoxColumn();
    c3.DataSource = getFormations();
    c3.HeaderText = "Formation";
    c3.DataPropertyName = "Formation"; //Propriété de Etudiant
    c3.AutoComplete = false;
    c3.DisplayMember = "Nom"; //Propriété de Formation
    c3.ValueMember = "Self"; //Propriété de Formation
    c3.MinimumWidth = 150;
    c3.AutoSizeMode = DataGridViewAutoSizeColumnMode.AllCells;

    dataGridViewEtudiants.Columns.Add(c1);
    dataGridViewEtudiants.Columns.Add(c2);
    dataGridViewEtudiants.Columns.Add(c3);
}
```

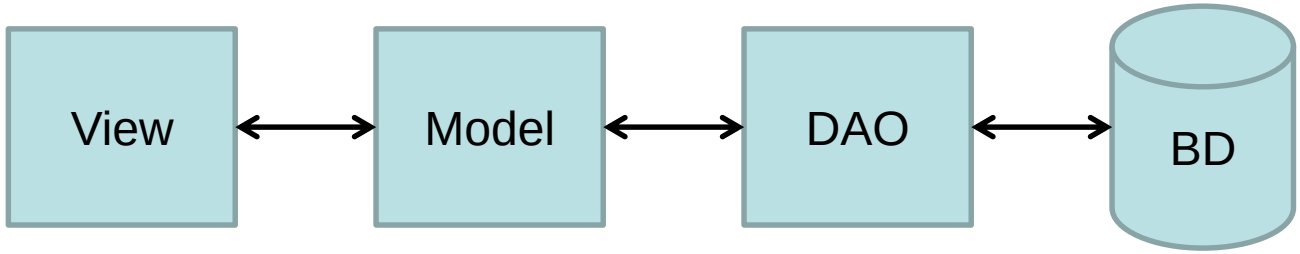
Nom	Prénom	Formation
DUPONT_1	Jean_1	Licence Pro
DUPONT_2	Jean_2	Licence Pro
DUPONT_3	Jean_3	Licence Pro
DUPONT_4	Jean_4	Licence Pro
DUPONT_5	Jean_5	Licence Pro
DUPONT_6	Jean_6	Licence Pro
DUPONT_7	Jean_7	Licence Pro
DUPONT_8	Jean_8	Licence Pro
DUPONT_9	Jean_9	Licence Pro
DUPONT_10	Jean_10	Licence Pro

A ajouter dans la classe Formation

```
public Formation Self
{
    get { return this; }
}
```

Mise en place du pattern DAO

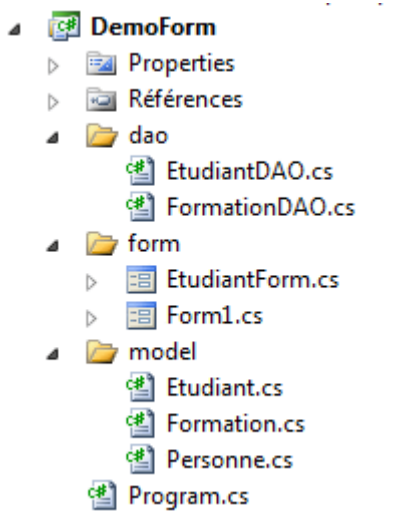
- Objet d'accès aux données
  - Objectif : Séparer l'accès aux données dans des objets dédiés



Exemple :	EtudiantForm	Etudiant	EtudiantDAO	Etudiants.xml

Mise en place du pattern DAO

- Organisation dans Visual Studio



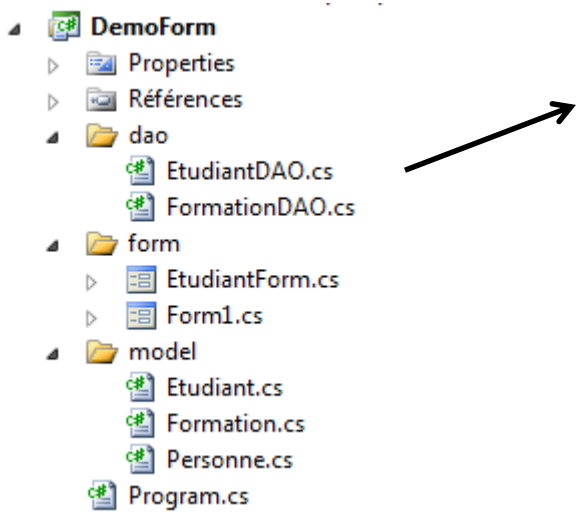
C'est là que vont être écrites les requêtes vers la base de données.
Les méthodes doivent retourner des objets métiers et non un curseur ou un enregistrement

Ce sont les fenêtres d'affichage et/ou de modification des données.
Elles ne doivent pas contenir de requête mais des appels aux objets DAO

Ce sont les objets métiers correspondants au modèle de la base de données.
Ils ne doivent être que des contenants d'informations et ne contenir que des méthodes d'accès ou de modifications des attributs

Mise en place du pattern DAO

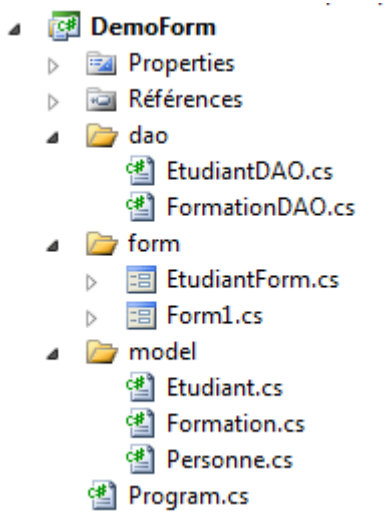
- Organisation dans Visual Studio




```
public List<Etudiant> getEtudiants()
{
    List<Etudiant> etudiants = null;
    XmlSerializer xs = new XmlSerializer(typeof(List<Etudiant>));
    StreamReader sr = null;
    try
    {
        sr = new StreamReader(ETUDIANT_XML);
        etudiants = xs.Deserialize(sr) as List<Etudiant>;
    }
    catch (Exception e)
    {
        Console.WriteLine("Problème lors de l'écriture du fichier xml: " + e.Message);
    }
    finally
    {
        sr.Close();
    }
    return etudiants;
}

public Etudiant getEtudiantByNumero(int num)
{
    Etudiant etudiant = null;
    foreach (Etudiant item in getEtudiants())
    {
        if (item.Numero.Equals(num))
        {
            etudiant = item;
            break;
        }
    }
    return etudiant;
}
```

Mise en place du pattern DAO

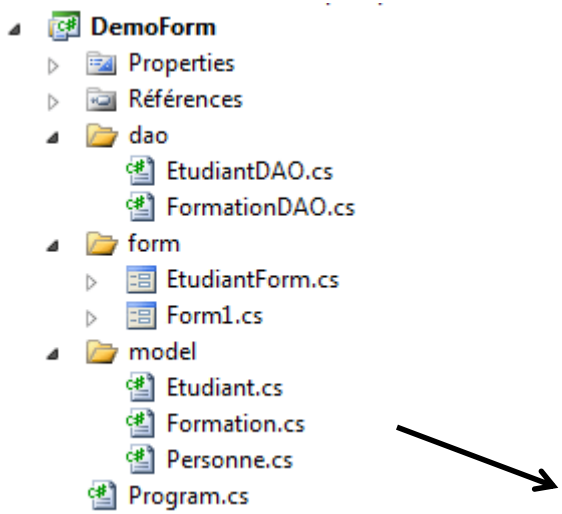
- Organisation dans Visual Studio




```
private void btnAfficher_Click(object sender, EventArgs e)
{
    int numero = int.Parse(textBoxNumero.Text);
    EtudiantDAO edao = new EtudiantDAO();
    Etudiant etudiant = edao.getEtudiantByNumero(numero);
    EtudiantForm ef = new EtudiantForm(etudiant);
    ef.ShowDialog();
}
```

Mise en place du pattern DAO

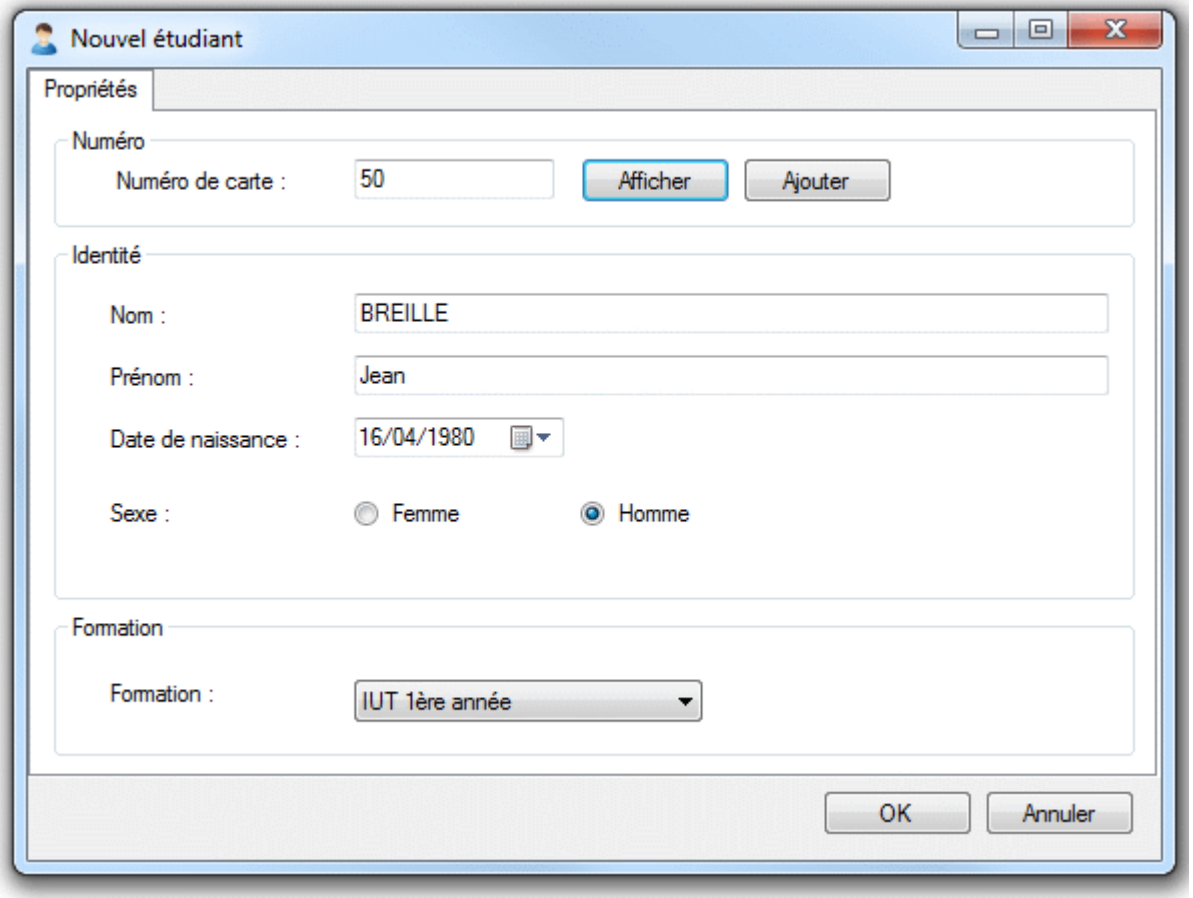
- Organisation dans Visual Studio




```
public class Etudiant : Personne
{
    private int numero;
    private Formation formation;
    private int anneeEntree;

    public Etudiant() { }

    public Etudiant(int id, string nom, string prenom, DateTime date, string sexe, int num, Formation formation)
    {
        base.Id = id;
        base.nom = nom; //utilisation des attributs de Personne
        base.prenom = prenom;
        base.DateNaissance = date;
        base.Sexe = sexe;
        this.numero = num;
        this.formation = formation;
    }

    public int Numero
    {
        get { return numero; }
        set { numero = value; }
    }

    public override bool Equals(object obj)
    {
        // si le paramètre ne peut pas être casté, on retourne false
        Etudiant e = obj as Etudiant;
        if ((object)e == null) return false;

        return this.numero.Equals(e.numero);
    }

    public override string ToString()
    {
        return this.prenom + " " + this.nom;
    }
```

Travaux pratiques – TP7

- Réalisation du programme ci-dessous



- Le bouton « Afficher » permet d'afficher les informations de l'étudiant dont le numéro a été saisi
- Le bouton « Ajouter » permet d'ajouter un nouvel étudiant à la liste
- Le bouton « OK », enregistre les modifications
- Le bouton « Annuler » ferme la fenêtre

Le tout en utilisant le pattern DAO