

ICT320

ANASS BENFARES &
XAVIER CARREL



ANASS BENFARES

POO

PROGRAMMATION ORIENTEE OBJET

○ Déroulement

- Accès à la Roadmap du cours :
<https://roadmap.sh/r/embed?id=66714f98c0f2325c34220bba>
- 5 périodes de cours avec moi et des périodes de projet avec M.Melly.
- Théorie puis exercices pratiques.
- Informations concernant les évaluations au prochain cours.
- Pas de questions bêtes !



○ Semaine 1 - Programmation orientée objet

- Programmation orientée objet c'est quoi ?
 - C'est un paradigme de programmation !
 - Organisation de notre code autour d'Objets !
 - On a donc besoin de :



- Mais surtout de:



○ Du Procédural à la POO

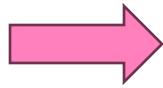
- Vous allez devoir passer d'un code qui inclut tout à un code où nous utiliserons **des classes** !
- Une classe qu'est-ce que c'est ?



○ Classes et Objets

- Vous allez devoir passer d'un code qui inclut tout (Procédural) à un code ou nous utiliserons **des classes** (POO)!
- Une classe qu'est-ce que c'est ?

Nos Objets



 informatique

○ Classes et Objets

- Les gâteaux, c'est bien mais on utilise ça comment ?
- Prenons l'exemple d'une manufacture:
 - Exemple usine Tesla !
 - On veut créer plusieurs types de voitures, mais on dispose de spécificités communes !
 - On va décrire ensemble les spécificités d'une voiture



○ Attributs

Une voiture dispose d'attributs -> Pourquoi et comment sait-on qu'une voiture est une voiture ?

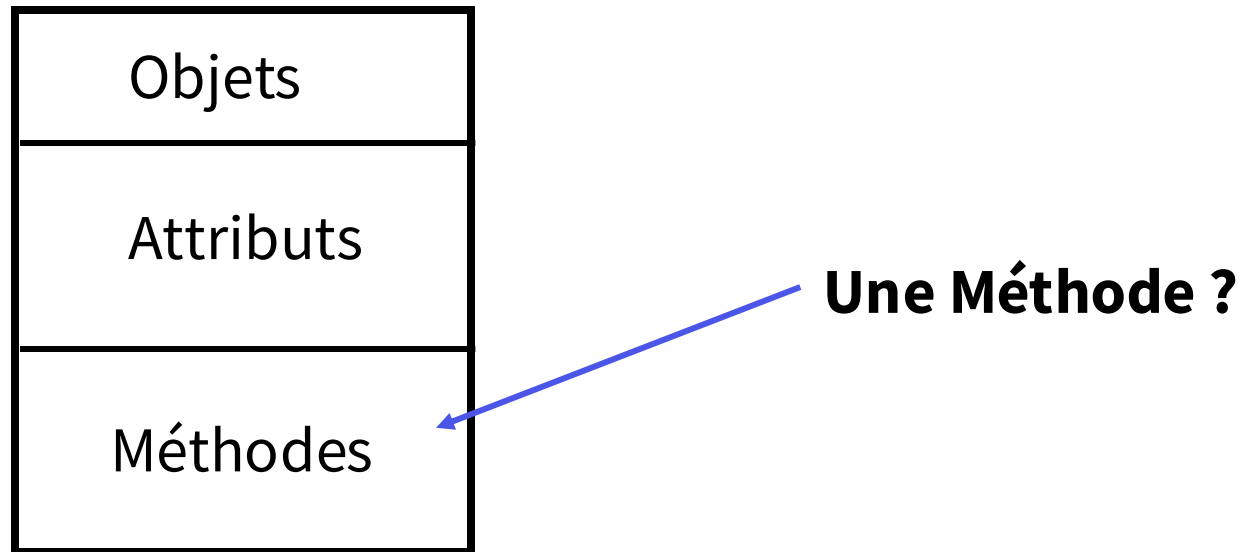
Une voiture dispose également de fonctionnalités !

On a comme attributs : Couleur, Nombre de portes,.. ?



○ Modélisation

Pour modéliser de la Programmation Orientée Objets on va utiliser les schémas UML.



○ Méthodes

Une méthode est une **Feature** de votre objet.

Par exemple pour la voiture elle peut :

- Démarrer
- Arrêter la voiture
- Allumer les phares

Possiblement aussi :

- Accélérer
- Freiner

Ces méthodes peuvent s'appeler entre elles.

Par exemple quand on démarre une voiture, l'allumage des phares est automatique !



○ Passons au code !

Pour créer un Objet, nous avons besoin d'un Constructeur dans notre classe !

C'est le point de départ de chaque objet.

On peut envoyer des arguments pour définir des paramètres liés aux spécificités.

Par exemple la couleur de ma voiture !





Ecrire une Class ?

```
class Car
{
    // Attributs
    private string _color;
    private int _doorNumber;

    /// <summary>
    /// Start a car
    /// </summary>
    public void start()
    {
        // TODO : create code
    }

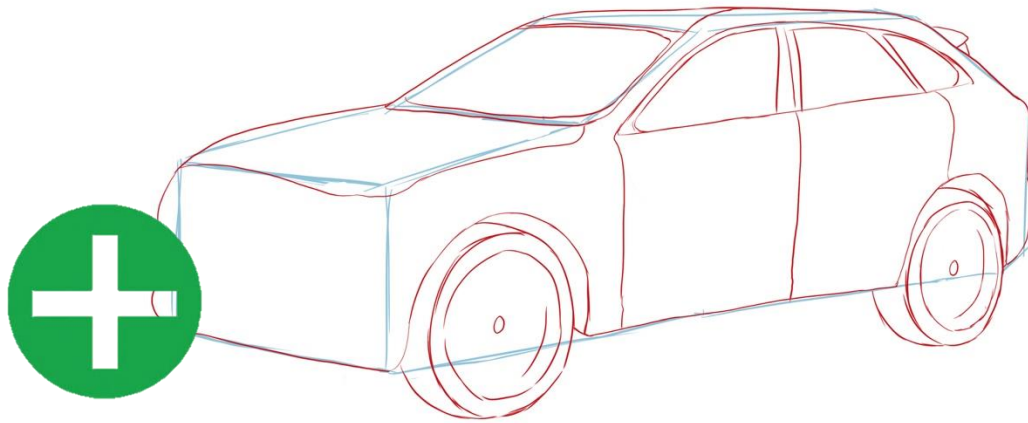
    /// <summary>
    /// Stop a car
    /// </summary>
    public void stop()
    {
        // TODO : create code
    }

    /// <summary>
    /// Turn on or turn off headlights for a car
    /// </summary>
    /// <param name="toggle">Turn on or turn off headlights</param>
    public void toggleHeadlights(bool toggle)
    {
        // TODO : create code
    }
}
```



○ Créer un objet / Constructeur

```
Car porsche = new Car();
```



○ Créer un objet / Constructeur

L'intérieur de la parenthèse, permet d'insérer des arguments

```
Car porsche = new Car();
```

porsche, est le nom de la variable de type objet Le **new**, va appeler le **constructeur**



Le **constructeur** est un **méthode spéciale** qui va permettre d'initialiser un objet et ses attributs lors de sa création.

Donc il initie les valeurs initiales.

Un class définit une structure, mais c'est un objet qui est créer !



Créer un objet / Visibilité

```
class Car
{
    // Attributs
    private string _color;
    private int _doorNumber;

    /// <summary>
    /// Start a car
    /// </summary>
    public void start()
    {
        // TODO : create code
    }

    /// <summary>
    /// Stop a car
    /// </summary>
    public void stop()
    {
        // TODO : create code
    }

    /// <summary>
    /// Turn on or turn off headlights for a car
    /// </summary>
    /// <param name="toggle">Turn on or turn off headlights</param>
    public void toggleHeadlights(bool toggle)
    {
        // TODO : create code
    }
}
```

Spécificité ?



Créer un objet / Visibilité

```
class Car
{
    // Attribut
    public string _color;
    private int _doorNumber;

    /// <summary>
    /// Start a car
    /// </summary>
    public void start()
    {
        // TODO : create code
    }

    /// <summary>
    /// Stop a car
    /// </summary>
    public void stop()
    {
        // TODO : create code
    }

    /// <summary>
    /// Turn on or turn off headlights for a car
    /// </summary>
    /// <param name="toggle">Turn on or turn off headlights</param>
    public void toggleHeadlights(bool toggle)
    {
        // TODO : create code
    }
}
```

Spécificité ?



○ Créer un objet / Visibilité

```
Car porsche = new Car();
```

On peut accéder à un attribut de la classe pour autant qu'il soit publique, de la sorte :

```
porsche._color = "Blue";
```

il est donc possible de modifier des éléments interne à la classe! ->  **Problème de Sécurité**



○ Encapsulation des données - Visibilité

Pour les méthodes et les attributs, il existe des niveaux de privilèges en fonction de la visibilité souhaitée.

Visibilité de la méthode / attribut	Influence
Private	Accessible depuis la class uniquement
Protected	Accessible depuis la class et les class qui héritent également (semaine 3)
Public	Accès depuis n'importe quelle autre partie du code, y compris à l'extérieur de la class ou du Namespace.





Encapsulation des données - Visibilité

```
public void start()
{
    this.toggleHeadlights(true);
    // TODO : create code
}

/// <summary>
/// Stop a car
/// </summary>
public void stop()
{
    toggleHeadlights(false);
    // TODO : create code
}

/// <summary>
/// Turn on or turn off headlights for a car
/// </summary>
/// <param name="toggle">Turn on or turn off headlights</param>
private void toggleHeadlights(bool toggle)
{
    // TODO : create code
}
```

Spécificité ?





Encapsulation des données - Visibilité

```
class Car
{
    // Properties
    private string _color;
    private int _doorNumber;

    /// <summary>
    /// Constructor
    /// </summary>
    /// <param name="color">color of car</param>
    /// <param name="doorNumber">number of door</param>
    public Car(string color, int doorNumber)
    {
        _color = color;
        _doorNumber = doorNumber;
    }

    /// <summary>
    /// Color property definition
    /// </summary>
    public string Color
    {
        get
        {
            return _color;
        }
        set
        {
            _color = value;
        }
    }

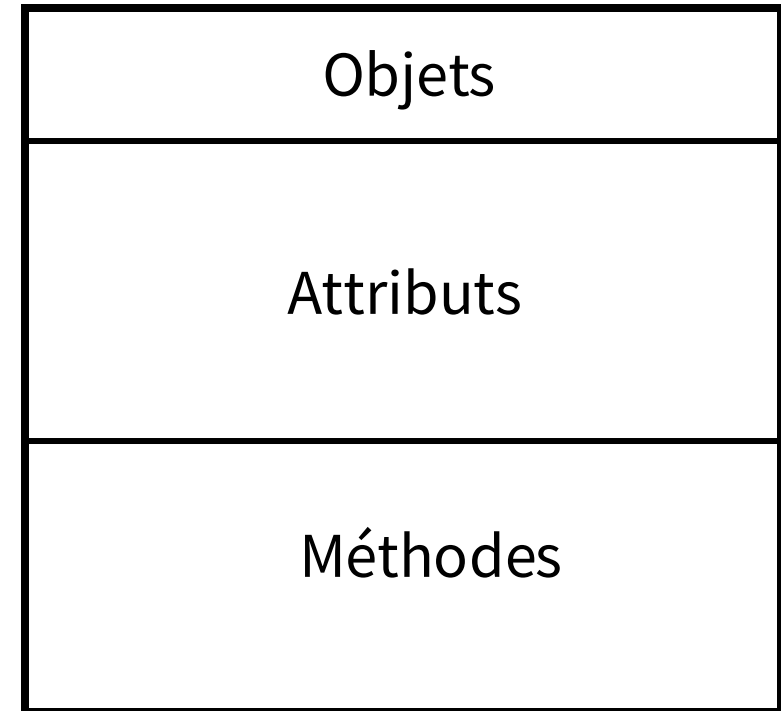
    /// <summary>
    /// DoorNumber property definition
    /// </summary>
    public string DoorNumber
    {
        get
        {
            return _doorNumber;
        }
        set
        {
            _doorNumber = value;
        }
    }
}
```



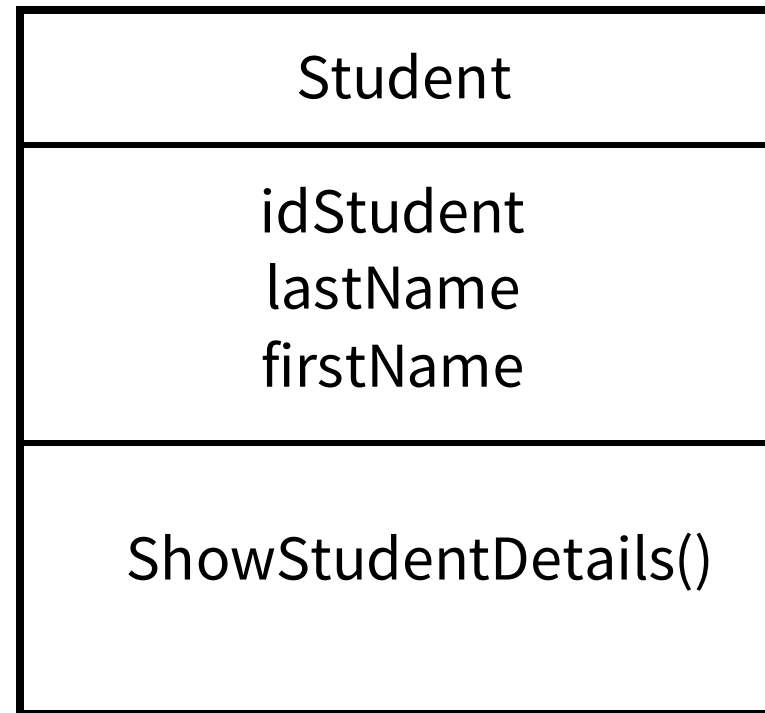
○ Class Eleve / Schéma UML

On aimerait créer des objets **Eleve**, qui vont permettre d'entrer un numéro d'étudiant, nom, prénom et d'afficher ces informations.

On aimerait que les informations sur l'élève ne puissent pas être accessible/modifier en dehors de la class !



○ Class Eleve / Schéma UML



Création d'une class Eleve

```
class Student
{
    // Attributs / Proprietes
    2 références
    private int _idStudent;
    2 références
    private string _firstName;
    2 références
    private string _lastName;

    /// <summary>
    /// Constructeur pour initialiser un nouvel élève
    /// <param name=idEleve'>Student ID</param>
    /// </summary>
    2 références
    public Student(int id, string prenom, string nom)
    {
        this._idStudent = id;
        this._lastName = nom;
        this._firstName = prenom;
    }

    // Méthode pour afficher les détails d'un élève
    2 références
    public void ShowStudentDetails()
    {
        Console.WriteLine("ID de l'élève : " + this._idStudent + " " + this._firstName + " " + this._lastName);
    }
}
```

```
0 références
static void Main()
{
    // Création de deux élèves
    Student eleve1 = new Student(1, "John", "Doe");
    Student eleve2 = new Student(2, "Jane", "Darc");

    // Affichage des détails des élèves
    eleve1.ShowStudentDetails(); // Affiche "ID de l'élève : 1"
    eleve2.ShowStudentDetails(); // Affiche "ID de l'élève : 2"
}
```

Création des objets dans le Main



○ Membres statiques

Les membres statiques sont des membres d'une class tout comme les attributs et les méthodes que nous avons déjà créé.

Ce sont des éléments qui sont **partagés** entre toutes les instances d'une classe sans avoir besoin de créer une instance dans cette classe.

Nous devons ajouté entre notre visibilité et notre nom de variable le terme « static ».



○ Membres statiques

Exemple :

Le nombre total d'étudiants est un attribut statique.

La méthode qui va afficher le nombre total d'étudiant est une méthode statique.





Membres statiques

```
class Student
{
    // Attributs / Propriétés
    2 références
    private int _idStudent;
    2 références
    private string _firstName;
    2 références
    private string _lastName;
    //Static attribut
    3 références
    public static int _studentTotal = 0;

    /// <summary>
    /// Constructeur pour initialiser un nouvel élève
    /// <param name=idEleve'>Student ID</param>
    /// </summary>
    2 références
    public Student(int id, string prenom, string nom)
    {
        this._idStudent = id;
        this._lastName = nom;
        this._firstName = prenom;
        _studentTotal++;
    }

    // Méthode statique pour afficher le nombre total d'élèves
    0 références
    public static void ShowStudentTotal()
    {
        Console.WriteLine("Nombre total d'élèves dans l'école : " + _studentTotal);
    }

    // Méthode non-statique pour afficher les détails d'un élève
    2 références
    public void ShowStudentDetails()
    {
        Console.WriteLine("ID de l'élève : " + this._idStudent + " " + this._firstName + " " + this._lastName);
    }
}
```

```
static void Main()
{
    // Création de deux élèves
    Student eleve1 = new Student(1, "John", "Doe");
    Student eleve2 = new Student(2, "Jane", "Darc");

    // Affichage des détails des élèves
    eleve1.ShowStudentDetails(); // Affiche "ID de l'élève : 1"
    eleve2.ShowStudentDetails(); // Affiche "ID de l'élève : 2"
    Console.WriteLine(Student._studentTotal);
}
```

Création des objets dans le Main



○ Membres statiques

Exemple :

Le nombre total d'étudiants est un attribut statique.

La méthode qui va afficher le nombre total d'étudiant est une méthode statique.



○ Autres Possibilités ?

Créer des objets Eleve et les mettre dans un objet Classe, exemple *Cin1B*.

Possibilité d'ajouter des **Namespace** également pour regrouper les class !



○ Collections de données

Dans de nombreux langages de programmation, il est possible de créer des collections de données.

C'est-à-dire une variable qui contiendrait plusieurs variables.

```
List<object> liste1 = new List<object>();

// Ajout de différents types d'éléments
liste1.Add(42);           // int
liste1.Add("Hello World"); // string
liste1.Add(3.14);         // double
liste1.Add(true);         // bool

// Affichage des éléments et de leur type
foreach (var variable in liste1)
{
    Console.WriteLine($"Valeur : {variable}, Type : {variable.GetType()}");
}
```



○ Collections de données

On pourrait par exemple créer une collection de données comme attribut d'une class et ajouter des objets d'une autre class dedans.

Essayons ensemble !



○ Collections de données

Nous avons également les tableaux en C# !

Contrairement aux listes, leur tailles sont fixes et ils consomment donc moins de mémoire.



○ Exercices pratiques

- Snail
- Parachutes

