

Langage C#

POO

Mme. Fatima-Ezzahra AIT BENNACER

f.aitbennacer@emsi.ma

2025 - 2026

Plan du cours

- 1 Introduction
- 2 Création d'une classe sous Visual Studio
- 3 Création d'un objet à partir d'une classe
- 4 Notion de visibilité
- 5 Création d'un objet/ Initialisation des attributs
- 6 ToString()
- 7 Accesseurs set et get
- 8 Les constructeurs
- 9 Attributs et méthodes statiques
- 10 Héritage

Introduction



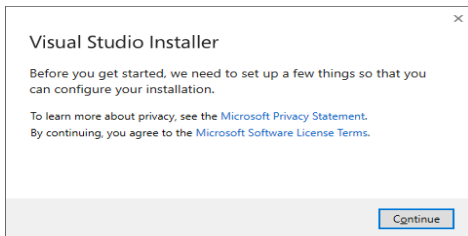
Introduction

- **Un objet:** représente un concept, une idée ou toute entité du monde physique comme une voiture, une personne ou encore un livre, etc.
- POO ou Programmation orientée Objet: Un paradigme de programmation informatique qui consiste en la définition et l'assemblage de briques logicielles appelées objet.

Installation de visual Studio

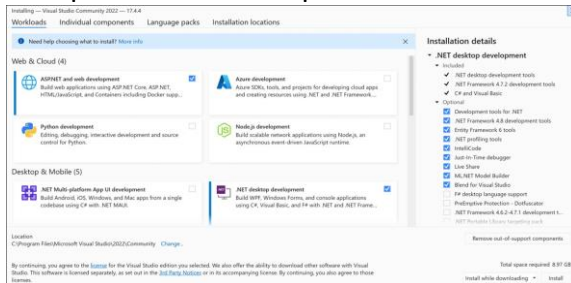
Installation de visual Studio

- **Étape 1:** Vous installez Visual Studio Community depuis <https://visualstudio.microsoft.com/fr/downloads/>
- **Étape 2:** Depuis votre dossier "Téléchargements", double-cliquez sur le programme d'installation.



Installation de visual Studio

- **Étape 3:** Une fois que Visual Studio Installer est installé, vous pouvez l'utiliser pour personnaliser votre installation. Lancez Visual Studio Installer, cliquez sur "Modifier", puis sélectionnez la charge de travail que vous souhaitez dans Visual Studio Installer.
- **Étape 4:** Vous sélectionnez .NET Desktop development & ASP.NET and web development et vous cliquez sur Install.



Notion de Classe

Classe

- Une classe est une manière de représenter un objet. En bref c'est la structure d'un objet.
- En C#, la définition des attributs et leurs getters/setters est assez simplifiée.
- La classe ne doit pas forcément avoir le même nom que le fichier.
- Dans un fichier, on peut définir plusieurs classes.

Classe

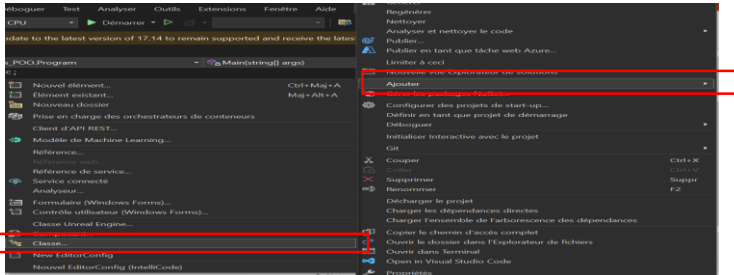
- C'est elle qui contenait la méthode spéciale **Main()** qui sert de point d'entrée à l'application.

Code : C#

```
class Program
{
    static void Main(string[] args)
    {
    }
}
```

Création d'une classe sous Visual Studio

- Faire un clic droit sur le nom du projet dans l'Explorateur de solutions.
- Aller dans Ajouter -> Class.
- Choisir Class.
- Saisir Voiture dans Nom : et valider.



Contenu de Voiture.cs

```
namespace MaPremiereApplication  
{  
  
    class Voiture  
    {  
  
    }  
  
}
```

Création d'un objet à partir d'une classe

Pour instancier une classe

- Le nom de la classe.
- Le nom de l'objet.
- L'opérateur new .
- Un constructeur de la classe.

Exemple:

```
static void Main(string[] args) {  
    Voiture voiture1 = new Voiture();  
    Voiture voiture2 = new Voiture();  
}
```

Pour Ajouter des attributs à une classe

Exemple

```
namespace MaPremiereApplication
{
    class Voiture

    {
        string Color;
        string Model ;
        int Price;
    }
}
```

Remarques

Il est impossible d'affecter des valeurs aux attributs de l'objet Voiture car la visibilité par défaut, en C#, est private

Notion de visibilité

Il existe plusieurs indicateurs de visibilité (peuvent être ajoutés avant attributs/classes/méthodes), mais les plus utilisés sont public et private:

Visibilité	Description
public	Accès non restreint
protected	Accès depuis la même classe ou depuis une classe dérivée
private	Accès uniquement depuis la même classe
internal	Accès restreint à la même assembly
protected internal	Accès restreint à la même assembly ou depuis une classe dérivée

Notion de visibilité

Pour accéder/affecter des valeurs aux attributs, on peut leur attribuer la visibilité public

Exemple

```
namespace MaPremiereApplication {  
class Voiture  
{  
    // Declaration des attributs  
    public string Color;  
    public string Model ;  
    public int Price;  
}  
}
```

Création d'un objet/ Initialisation des attributs

Affichage de l'objet voiture

```
static void Main(string[] args)
{
    Voiture voiture = new Voiture();
    voiture.Color = "red";
    voiture.Model = "Toyota";
    voiture.Price = 167000;
    Console.WriteLine(voiture);
    Console.ReadKey();
}
```

Création d'un objet/ Initialisation des attributs

Affichage des valeurs des attributs de l'objet voiture

```
static void Main(string[] args)
{
    Voiture voiture = new Voiture();
    voiture.Color = "red";
    voiture.Model = "Toyota";
    voiture.Price = 167000;
    Console.WriteLine($"Les caractéristiques sont: {
    voiture.Color } { voiture.Model }");
    Console.ReadKey();
}
```

ToString()

Code généré de la méthode ToString()

```
public override string ToString()  
{  
    return base.ToString();  
}
```

Affichage des détails d'un objet: Modification du contenu de ToString()

```
public override string ToString()  
{  
    return "Voiture [Color=" + Color + ", Model=" + Model + ",  
    Price=" + Price + "]";  
}
```

Accesseurs set et get

Exemple

```
class Person {  
    private string name;  
    public string Name {get => name;  
        set => name = value;  
    }  
}
```

Dans Main

```
Person person = new Person();  
person.Name = "Sara"; // the set accessor is invoked here  
Console.WriteLine($"Je m'appelle { personne.Name}");  
// the get accessor is invoked here
```

Accesseurs set et get

Si les getters et setters ne contiennent pas un traitement particulier, on peut supprimer les attributs et remplacer les getters et setters précédents par les suivants

```
public class Person {  
    public string Name { get; set;}  
}
```

Rien ne change pour l'appel

Accesseurs set et get

Exercice 1

Définir une classe `Personne` avec trois propriétés (Nom, Prénom et Age).

L'Age doit être supérieur à 0 et inférieur à 100

Accesseurs set et get

En supprimant le set : l'Age devient accessible seulement en lecture

```
public class Personne {  
    private int age  
    public int Age { get; }  
}
```

En supprimant le get : l'Age devient accessible seulement en écriture

```
public class Personne {  
    private int age  
    public int Age { set; }  
}
```

Le constructeur

- Une méthode particulière portant le nom de la classe et ne retournant aucune valeur.
- Toute classe en C# a un constructeur par défaut sans paramètre.
- Ce constructeur sans paramètre n'a aucun code.
- On peut le définir si un traitement est nécessaire.
- La déclaration d'un objet de la classe fait appel à ce constructeur sans paramètre.
- Toutefois, et pour simplifier la création d'objets, on peut définir un nouveau constructeur qui prend en paramètre plusieurs attributs

Le constructeur

Exemple

```
public class CompteBancaire
{
    private string titulaire, devise;
    private double solde;
    // Constructeur
    public CompteBancaire(string Titulaire, double Solde, string Devise)
    {
        this.titulaire = Titulaire;
        this.solde = Solde;
        this.devise = Devise;
    }
}
```

Le constructeur

Dans Main

```
static void Main(string[] args) {  
    CompteBancaire compte = new CompteBancaire("Jad", 1700000,  
    "Euro" );  
    Console.WriteLine(compte);  
    CompteBancaire compte2 = new CompteBancaire("Rhizlane",  
    200000, "MAD");  
    Console.WriteLine(compte2);  
}
```

En définissant ce constructeur avec trois paramètres, le constructeur par défaut (sans paramètre) n'existe plus, le Main ne peut être exécuté.

Le constructeur

Solution

```
public class CompteBancaire
{
    public string titulaire, devise;
    public double solde;
    public CompteBancaire(string Titulaire, double Solde, string Devise)
    {
        this.titulaire = Titulaire;
        this.solde = Solde;
        this.devise = Devise;
    }
    public CompteBancaire(){
    }
}
```

Le constructeur

Dans Main

```
static void Main(string[] args) {  
    CompteBancaire compte = new CompteBancaire("Jad", 1700000,  
    "Euro" );  
    Console.WriteLine(compte.ToString());  
    Personne compte2 = new CompteBancaire("Rhizlane", 200000,  
    "MAD");  
    Console.WriteLine(compte2.ToString());  
}
```

Attributs et méthodes statiques

Attributs et méthodes statiques

Les instances d'une même classe ont toutes les mêmes attributs mais pas les mêmes valeurs.

Si nous désirions qu'un attribut ait une valeur partagée par toutes les instances (le nombre d'objets instanciés de la classe *Personne*), Qu'est ce qu'on doit faire?

⇒ **Attribut statique ou attribut de classe**

Attribut statique ou attribut de classe: Un attribut dont la valeur est partagée par toutes les instances de la classe est appelée.

Attributs et méthodes statiques

Si on veut créer un attribut contenant le nombre des objets créés à partir de la classe *Personne*.

Notre attribut doit être *static*, sinon chaque objet pourrait avoir sa propre valeur pour cet attribut.

Ajoutons un attribut *static* appelé **NbrPersonnes** dans la classe *Personne*

```
public static int NbrPersonnes { get; set; }
```

Attributs et méthodes statiques

Incrémentons la valeur de **NbrPersonnes** dans les différents constructeurs de la classe **Personne**

```
public Personne(int num, string name) {  
    this.num = Num;  
    this.name = Name;  
    NbrPersonnes++;  
}  
public Personne() {  
    NbrPersonnes++;  
}
```

Attributs et méthodes statiques

Testons cela dans le Main

```
static void Main(string[] args) {  
    Console.WriteLine(Personne.NbrPersonnes);  
  
    Personne personne1 = new Personne(200, "Ahmed Ahmed");  
    Console.WriteLine(Personne.NbrPersonnes);  
    Console.WriteLine(personne1);  
}
```

Exercices POO

Exercice 2

Créer un constructeur **Cercle** qui crée un cercle avec un rayon fourni par un argument.

Les cercles construits doivent avoir deux méthodes **GetAire()** et **GetPerimetre()** qui donnent à la fois l'aire et le périmètre respectifs.

Héritage

L'héritage

- Lorsque deux ou plusieurs classes partagent plusieurs attributs (et méthodes).
- Consiste à créer une nouvelle classe dite classe dérivée ou classe fille à partir d'une classe existante dite classe de base ou classe parente ou classe mère.

L'héritage

L'héritage permet de :

- Récupérer le comportement standard d'une classe objet (classe parente) à partir de propriétés et de méthodes définies dans celle-ci.
- Ajouter des fonctionnalités supplémentaires en créant de nouvelles propriétés et méthodes dans la classe dérivée.
- Modifier le comportement standard d'une classe d'objet (classe parente), en surchargeant certaines méthodes de la classe parente dans la classe dérivée.

L'héritage

Exemple

- ❑ Un Enseignant a un ID, un nom, un prénom et un salaire.
- ❑ Un Etudiant a aussi un ID, un nom, un prénom et une note.
- ❑ Sémantiquement, Enseignant et Etudiant sont une sorte de Personne.
- ❑ En plus, les deux partagent plusieurs attributs tels que ID, nom et prénom.
- ❑ Donc, on peut mettre en commun les attributs ID, nom et prénom dans une classe Personne.
- ❑ Les classes Etudiant et Enseignant hériteront de la classe Personne.

L'héritage

Classe parente

```
class ClassA {  
    //propriété de la classe  
    public int DataA ;  
    //Méthode de la classe  
    public int FonctionA1(){  
        //code de la fonction fonctionA1  
    }  
    public virtual int FonctionA2() //redéfinissable  
    {  
        //code de la méthode fonctionA2  
    }  
}
```

L'héritage

Classe fille

```
class ClassB : ClassA {  
    //propriété de la classe  
    public int DataB;  
    //Méthode de la classe  
    public override int FonctionA2(){  
        //code de la fonction fonctionA2  
    }  
    public int FonctionB1() //redéfinissable  
    {  
        //code de la méthode fonctionB1  
    }  
}
```

L'héritage

- ❑ dataA est une propriété de la classe ClasseA. Par héritage dataA est aussi une propriété de la classe ClasseB.
- ❑ dataB est une propriété de la classe ClasseB (mais pas de la classe ClasseA).
- ❑ FonctionA1 est une méthode de la classe ClasseA. Par héritage FonctionA1 est aussi une méthode de la classe ClasseB.
- ❑ FonctionB1 est une méthode de la classe ClasseB (mais pas de la classe ClasseA).
- ❑ FonctionA2 est une méthode des classes ClasseA et ClasseB.
 - ❑ Dans la classe ClasseA, FonctionA2() est déclarée virtual car elle est redéfinissable dans la classe ClasseB.
 - ❑ Dans la classe ClasseB, FonctionA2() est déclarée override car elle remplace la méthode de la classe ClasseA.

L'héritage

Classe Personne

```
namespace MonProjet {  
class Personne  
{  
    public int Num { get; set; }  
    public string Nom { get; set; }  
    public string Prenom { get; set; }  
}  
}
```

Classe Enseignant

```
namespace MonProjet {  
class Enseignant : Personne  
    { public int Salaire { get; set; } }  
}
```

L'héritage

Classe Etudiant

```
namespace MonProjet {  
class Etudiant : Personne  
{  
    public string Niveau { get; set; }  
}  
}
```

L'héritage

Pour créer un objet de type Enseignant

```
namespace MonProjet {  
    Enseignant enseignant = new Enseignant();  
    enseignant.Num = 10;  
    enseignant.Nom = "MonNom";  
    enseignant.Prenom = "MonPrenom";  
    enseignant.Salaire = 15000;  
    Console.WriteLine(enseignant.ToString());  
}
```

On ne voit pas le salaire, pourquoi ?

⇒ Comme on n'a pas redéfini la méthode ToString(), on a utilisé celle de la classe mère

L'héritage

Ajoutons **ToString()** dans la classe Enseignant

```
namespace MonProjet {  
    public override string ToString() {  
        return base.ToString() + " Enseignant [salaire=" + Salaire + "]; }  
}
```

Ajoutons **ToString()** dans la classe Etudiant

```
public override string ToString() {  
    return base.ToString() + " Etudiant [niveau=" + Niveau + "]; }  
}
```

Remarque

⇒ Le mot-clé *base* permet d'appeler une méthode de la classe mère.

L'héritage

La classe Enseignant avec un constructeur à quatre paramètres et ToString

```
class Enseignant : Personne
{
    public Enseignant(int num, string nom, string prenom, int
    salaire) : base(num, nom, prenom) { Salaire = salaire; }
    public Enseignant() { }
    public int Salaire { get; set; }
    public override string ToString() { return base.ToString() + "
    Enseignant [salaire=" + Salaire + "]; }
}
```

L'héritage

La classe Etudiant avec un constructeur à quatre paramètres et ToString

```
class Etudiant : Personne
{
    public Enseignant(int num, string nom, string prenom, string
niveau) : base(num, nom, prenom) { Niveau = niveau; }
    public Etudiant() { }
    public string Niveau { get; set; }
    public override string ToString()
    {
        return base.ToString() + " Etudiant [niveau=" + Niveau + "]; }
}
```

L'héritage

Un objet de la classe `Personne` peut être créé

```
Enseignant enseignant2 = new Enseignant(4, "Adib", "Abdellah",  
40000);
```

Ou

```
Personne enseignant2 = new Enseignant(4, "Adib", "Abdellah",  
40000);
```

À ne pas faire!

```
Enseignant enseignant2 = new Personne(4, "Adib", "Abdellah",  
40000);
```

L'héritage

Remarques

Pour connaître la classe d'un objet, on peut utiliser le mot-clé **is**

Exemples

```
Console.WriteLine(enseignant2 is Enseignant);  
// affiche True  
Console.WriteLine(enseignant2 is Personne);  
// affiche True  
Console.WriteLine(personne is Enseignant);  
// affiche False
```

TP N°2