ENCAPSULATION ET MASQUAGE DE L'INFORMATION

Par **Sillery TALLA.**
Enseignant Ing es Informatique .GL

Programmation Evènementielle et IHM2

Table des matières

Objectifs de la leçon
- qu'es ce que l'encapsulation ? comment implémente ton le masquage de l'information ? quel est son utilité dans un concept de POO ??

Prérequis
avoir fait la programmation évènementielle I et avoir des connaissance avérée en Algorithme et en Informatique fondamental

I. LE CONCEPT D'ENCAPSULATION

L'encapsulation est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet, c'est-à-dire en empêchant l'accès aux données par un autre moyen que les services proposés. L'encapsulation permet donc de garantir l'intégrité des données contenues dans l'objet.

Si l'on veut protéger des informations contre une modification inattendue, on doit se référer au principe d'encapsulation. Il est également utilisé, peut-être la plupart de cas inconsciemment, par des développeurs PHP.

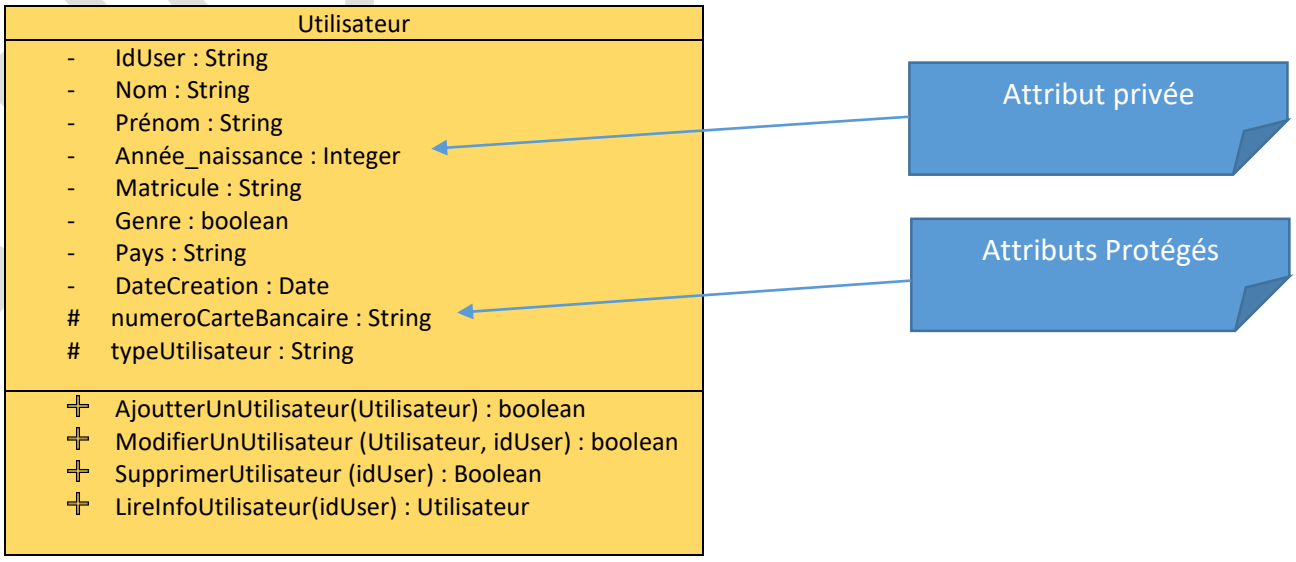
- Doit-on voir / accéder à tous les attributs ou à toutes les opérations d'un objet ? Non. La classe définit ce qui est visible / accessible. C'est le principe de l'encapsulation. Un objet complexe ne peut être utilisé qu'au travers de ce qui est accessible

Analogie :

- ☛ Il n'est possible d'utiliser une voiture qu'à travers son volant, son frein, son accélérateur, etc.
- ☛ L'accès au carburateur est impossible sauf par les méthodes qui le font de manière cohérente (méthode accélérer de l'accélérateur)

II. NOTATION UML

La programmation Orienté Objet s'appuie évidement sur l'analyse objet (UML) que vous verrez en profondeur lors de votre UV qui vous sera fait certainement plus tard ou que vous avez déjà fait certainement. Ainsi le concept d'encapsulation trouve donc bien évidement sa notation en UML comme l'indique l'exemple ci-dessous.



Cours par Sillery Philinte TALLA.
ENS, ING es d'informatique : Génie Logiciel
tel : 691748658 Email : tsillery@ymail.com

III. QU'EST-CE QUE L'ENCAPSULATION EN VISUAL BASIC ?

Le principe d'encapsulation consiste sur la protection des informations contenues dans un objet. Cette protection peut être représentée par l'absence de possibilité de modifier ces données depuis l'extérieur. On peut accéder à ces informations uniquement par les services proposés.

```
Public Class Utilisateur

    Public Nom As String
    Public Prenom As String
    Public Annee_naissance As Integer
    Public Matricule As String
    Public Genre As String
    Public Pays As String
    Public DateCreation As Date
    Public NumeroCarteBancaire As String
    Public typeutilisateur As String

    1 référence
    Sub New()
        Console.WriteLine("Nous somme dans le Constructeur ")
    End Sub


End Class
```

Bavardage :

- Nous avons ci-dessus un exemple de classe Utilisateur non encapsulé (c'est-à-dire qui a des donnée (attributs/propriétés) non protégé.
- Faut aussi noter que lorsque le modificateur n'est pas spécifié il prend implicitement la valeur **private.** Ce qui engendre une encapsulation partielle car ne prévoit pas les méthode d'accecibilité au attribut ... Voici ci-dessous l'exemple d'une classe encapsulé. Dans l'exemple ci-dessous nous allons implémenter comme décrite dans le schéma UML proposé plus haut.

```
3 références
Public Class Utilisateur

    Private Nom As String
    Private Prenom As String
    Private Annee_naissance As Integer
    Private Matricule As String
    Private Genre As String
    Private Pays As String
    Private DateCreation As Date
    Private NumeroCarteBancaire As String
    Private typeutilisateur As String

    1 référence
    Sub New()
        Console.WriteLine("Nous somme dans le Constructeur ")
    End Sub


End Class
```

Bavardage

- La première remarque est que lorsque vous utilisez VS 2019 ou autre version, dès que vous avez changé l'attribut en **Private** alors la coloration syntaxique change
- Maintenant la classe est encapsulée (donnée protégé) mais aucune méthode ne permet d'accéder à ces attributs.

Nous allons maintenant créer de manière des méthode d'accès a des attributs des classes. Deux manière de faire son possible. On a la possibilité de créer des méthodes normales (fonction et procédures) qui auront la responsabilité de jouer ce rôle ou encore utiliser les méthodes natives d'accès a des propriétés qui auront pour rôle de jouer ce rôle. Pour l'instant nous allons nous intéresser à la deuxième façon de faire afin de couler dans le langage lui-même. On les appelle les Accesseurs.

Les accesseurs sont de deux types. Les accesseurs en lectures et les Accesseurs en écriture. Dans la plupart des langages on les appelle les **Getters et les Setters.** Leur utilisation est implicitement définie dans leur appellation. Les Getters permettent de récupérer / lire la valeur de l'élément auquel il est associé tandis que les Setters permet de modifier la valeur de l'élément auquel il est associé.

- **LES GETTERS en VISUAL BASIC .NET**

```
    Private _nom As String
    1 référence
    Public Property Nom As String
        Get
            Return _nom
        End Get
```

Définir la propriété Get d'un attribut en Visual Basic, il faut au préalable définir la propriété qui lui est associé et déclarer par la suite la fonction native **Get** pour retourner la valeur

- A la ligne 3 on déclare normalement le nom comme propriété privée via le **Private**
- La ligne 4 permet de déclarer la propriété Function Accesseur pour y accéder. On note la présence de
    - Le modificateur **Public** pour dire qu'il sera acceccible a l'extérieur de la classe.
    - Et le mot clé **Property qui en VB.NET** marque la déclaration d'un Accesseur
    - Dans la fonction **Get** on implémentera toutes les conditions d'accecibilité qu'on aura définit dans notre conception.

☛ LES SETTERS EN VB.NET

```
3       Private _nom As String
        1 référence
4       Public Property Nom As String
5           Get
6               Return _nom
7           End Get
8           Set(value As String)
9               _nom = value
10          End Set
11      End Property
```

- Le Set permet de modifier une Valeur d'une propriété
- Il permet de définir aussi les conditions de modifications d'un attribut en fonction de la conception qu'on aura faite de notre programme.

Ci-dessous un Exemple de début d'implémentation de l'encapsulation des éléments d'une classe en VB.NET.

```
3       Private _nom As String
        1 référence
4       Public Property Nom As String
5           Get
6               Return _nom
7           End Get
8           Set(value As String)
9               _nom = value
10          End Set
11      End Property
12
13      Private _prenom As String
        0 références
14      Public Property Prenom As String
15          Get
16              Return _prenom
17          End Get
18          Set(value As String)
19              _prenom = value
20          End Set
21      End Property
```

IV. LES AVANTAGES DE L'ENCAPSULATION

L'encapsulation :

- garantit une meilleure flexibilité du code
- garantit un meilleure contrôle les données (niveaux d'accès)
- aide à concevoir des objets immuables
- permet de mieux gérer les parties du code

V. MASQUAGE DE L'INFORMATION

L'utilisateur d'une classe n'a pas forcément à savoir de quelle façon sont structurées les données dans l'objet, cela signifie qu'un utilisateur n'a pas à connaître l'implémentation. Ainsi, en interdisant l'utilisateur de modifier directement les attributs, et en l'obligeant à utiliser les fonctions définies pour les modifier (appelées interfaces), on est capable de s'assurer de l'intégrité des données (on pourra par exemple s'assurer que le type des données fournies est conforme à nos attentes, ou encore que les données se trouvent bien dans l'intervalle attendu).

L'encapsulation permet de définir des niveaux de visibilité des éléments de la classe. Ces niveaux de visibilité définissent les droits d'accès aux données selon que l'on y accède par une méthode de la classe elle-même, d'une classe héritière, ou bien d'une classe quelconque. Il existe trois niveaux de visibilité :

- **Publique / Public** : les fonctions de toutes les classes peuvent accéder aux données ou aux méthodes d'une classe définie avec le niveau de visibilité public. Il s'agit du plus bas niveau de protection des données

- **Protégée / Protected** : l'accès aux données est réservé aux fonctions des classes héritières, c'est-à-dire par les fonctions membres de la classe ainsi que des classes dérivées

- **Privée** / **Private** : l'accès aux données est limité aux méthodes de la classe elle-même. Il s'agit du niveau de protection des données le plus élevé

- **Friend** : aucun autre projet ne pourra avoir accès a cette donnée.