Chapitre 2 – Attributs et visibilité

Au Chapitre 1, n'importe qui pouvait écrire `voiture.NombreDePortes = -12` sans que personne ne proteste. Le compilateur accepte, l'objet se retrouve dans un état impossible. C'est le signe que nos attributs ne sont pas protégés. Dans ce chapitre, on découvre les deux mots-clés qui posent les bases de la solution : `public` et `private`.

5TTR 6TTR Découverte

La fin du Chapitre 1 a laissé un trou : `voiture.NombreDePortes = -12` est accepté sans broncher. Ce chapitre regarde ce trou de plus près, et pose la première pierre pour le boucher.

🎯 Objectifs

À la fin de ce chapitre, tu seras capable de :

1. Expliquer pourquoi des attributs accessibles à tous posent problème.
2. Utiliser les mots-clés `public` et `private` correctement.
3. Reconnaître l'erreur de compilation qui apparaît quand on tente d'accéder à un attribut privé depuis l'extérieur.
4. Énoncer la règle de l'encapsulation : *attributs privés, accès contrôlé*.

Partie 1 — Un objet sans défense

Reprends la classe `Voiture` du Chapitre 1 :

```
class Voiture
{
    public string Marque;
    public string Couleur;
    public int NombreDePortes;
}
```

Et son utilisation :

```
Voiture maVoiture = new Voiture();
maVoiture.Marque = "Peugeot";
maVoiture.NombreDePortes = 5;     // Cohérent.
```

```
maVoiture.NombreDePortes = -12;   // Absurde. Mais accepté.
maVoiture.NombreDePortes = 9999;  // N'importe quoi. Accepté aussi.
```

Le compilateur ne bronche pas. À l'exécution, rien n'explose. **L'objet est dans un état impossible, et personne ne le sait.**

Ce n'est pas spécifique à `Voiture`. Le même problème touche n'importe quelle classe avec des attributs `public` :

```
Personnage hero = new Personnage();
hero.Pv     = -9999;   // Personne ne proteste.
hero.Niveau = -42;     // Idem.
```

> 🧠 **L'IDÉE À SAISIR :** la responsabilité de protéger les données d'un objet doit être **à l'intérieur de cet objet**. C'est à l'objet de refuser ce qui n'a pas de sens — pas au code appelant de penser à vérifier à chaque ligne.

Pour ça, il faut commencer par **fermer la porte de l'extérieur**. C'est le rôle de la visibilité.

Partie 2 — Les mots-clés `public` et `private`

En C#, chaque membre d'une classe (attribut, méthode, propriété…) a une **visibilité**. Elle décide d'où ce membre peut être lu ou modifié.

Deux mots-clés à connaître pour démarrer :

Mot-clé	Accessible depuis	Usage typique
`public`	N'importe où dans le projet	Méthodes, propriétés (l'interface de l'objet)
`private`	Uniquement à l'intérieur de la classe	Attributs internes (les données privées)

> 💡 **SI ON NE MET RIEN**, C# considère le membre comme `private` par défaut. On préfère quand même l'écrire explicitement — c'est plus lisible.

Rendons les attributs de `Voiture` privés

```
class Voiture
{
    private string marque;          // ← private
```

```
    private string couleur;         // ← private
    private int    nombreDePortes;  // ← private
}
```

ℹ️ **CONVENTION DE NOMMAGE** : les attributs privés s'écrivent traditionnellement en *camelCase* (première lettre minuscule) : `nombreDePortes`. Les propriétés et méthodes publiques s'écrivent en *PascalCase* (première lettre majuscule) : `NombreDePortes`, `Demarrer()`. On verra l'intérêt de cette distinction au chapitre suivant.

Partie 3 — Conséquence : le code d'avant ne compile plus

Reprends maintenant ton `Program.cs` du Chapitre 1 :

```
Voiture maVoiture = new Voiture();
maVoiture.marque = "Peugeot";       // ❌ Erreur de compilation
maVoiture.couleur = "Rouge";        // ❌
maVoiture.nombreDePortes = 5;       // ❌
```

Dans Rider, tu vois apparaître une erreur en rouge sur chacune de ces lignes :

```
'Voiture.marque' is inaccessible due to its protection level
```

C'est exactement ce qu'on voulait. Le compilateur empêche maintenant n'importe qui de modifier les données de la voiture en passant par-dessus la classe.

Sauf que… on a un nouveau problème : **on ne peut plus rien faire avec notre voiture**. On a fermé la porte, mais on n'a pas encore la clé.

⚠️ **ERREUR CLASSIQUE** : passer en `private` *sans* prévoir comment l'extérieur va lire et écrire les données. La classe devient inutilisable. Si tu casses tout ton `Program.cs` en mettant `private` partout, c'est normal — la solution est dans les deux chapitres suivants.

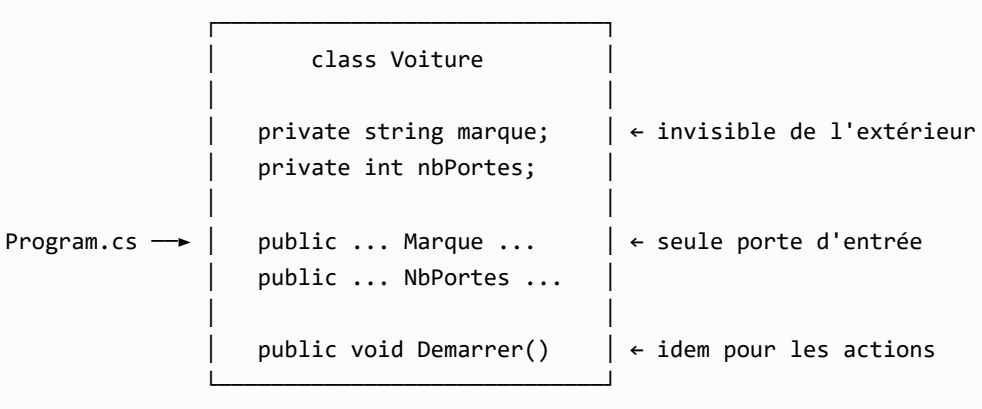
Partie 4 — La règle de l'encapsulation

L'encapsulation, c'est l'idée que **les données d'un objet sont cachées à l'intérieur de cet objet, et qu'on n'y accède que par des points d'entrée contrôlés**.

La règle de base, à retenir une fois pour toutes :

> 🧠 **LES ATTRIBUTS SONT `PRIVATE`. LES POINTS D'ACCÈS SONT `PUBLIC`.**

Schématiquement :



Tout ce qui dépasse à l'extérieur de la classe, c'est **l'interface publique** de l'objet. C'est elle qui décide ce qu'on peut faire avec lui, et dans quelles limites.

Les méthodes `Demarrer()`, `AfficherInfos()` du Chapitre 1 sont déjà `public` — c'est normal, elles sont faites pour être appelées depuis l'extérieur. Ce qui change ici, c'est que **les données** elles, deviennent `private`.

🧪 Exercices

Exercice 1 — Casser et observer

Reprends ton projet du Chapitre 1. Passe **tous les attributs de `Voiture` en `private`**, puis essaie de lancer le programme.

1. Note les erreurs de compilation affichées par Rider.
2. Compte combien de lignes de `Program.cs` sont concernées.
3. Conclus : pourquoi est-ce une **bonne nouvelle**, même si le programme ne tourne plus pour l'instant ?

Exercice 2 — Identifier les attributs à protéger

Pour chacune des classes ci-dessous, indique :

- Quels attributs devraient être `private` (presque toujours : tous).
- Pour chaque attribut, donne **un exemple de valeur absurde** qui devrait être refusée si quelqu'un essayait de l'affecter directement.

Classe	Attributs
Personnage	`nom` , `pv` , `pvMax` , `niveau`
CompteBancaire	`titulaire` , `solde`
LampeConnectee	`couleur` , `intensite` (0 à 100), `allumee`
Produit	`nom` , `prix` , `quantiteEnStock`

Exercice 3 — Lecture d'erreur

Tu écris ce code dans Rider :

```
class Compte
{
    private double solde;
}

// Dans Program.cs :
Compte c = new Compte();
c.solde = 1000;
Console.WriteLine(c.solde);
```

Rider affiche une erreur. **Sans la lire** :

1. Écris la phrase d'erreur que tu **devines** voir apparaître.
2. Sur quelles lignes apparaît-elle ?
3. Compile et compare avec ce que Rider montre réellement.

✍️ À retenir

- Des attributs `public` exposent un objet à des **affectations absurdes** que rien n'empêche.
- `public` = accessible partout · `private` = accessible uniquement dans la classe.
- La règle de l'encapsulation : **attributs** `private` , **interface** `public` .
- Mettre `private` casse temporairement le code appelant — c'est attendu. Il faut maintenant **ouvrir des points d'accès contrôlés** : c'est l'objet des chapitres 3 et 4.

Suite

On a fermé la porte. Maintenant, il faut une serrure et une clé : un moyen de **lire et d'écrire** ces attributs privés depuis l'extérieur, mais **en gardant le contrôle**. La solution la plus classique, qu'on rencontre dans tous les

langages objet (Java, PHP, Python…), s'appelle les **getters** et les **setters**. C'est l'objet du chapitre suivant.

© 2025 Mr. Lambrechts https://v2.ttrinfo.be 6 | 6