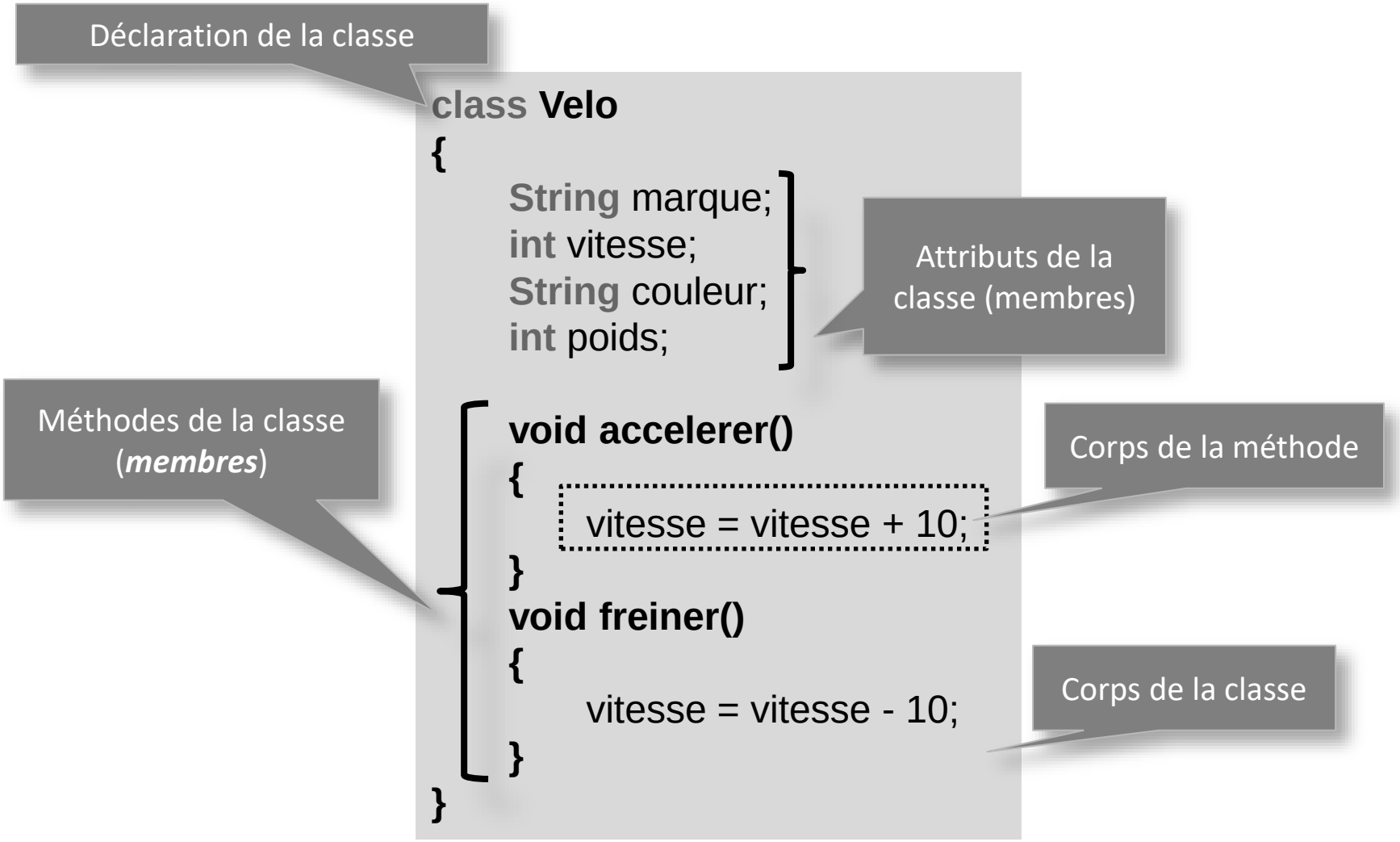
Orienté Objet :
Les classes en C

La classe en C#

- Comme nous l'avons vu au chapitre précédent, une classe va nous permettre de créer des objets…mais avant cela, nous devons apprendre à définir une classe de manière à ce que les objets qui seront instanciés correspondent à ce que nous avions imaginé.

- L'écriture d'une classe en C# est soumise à des règles et utilise certains des mots clé du langage.

Une première classe – Structure de base



Membres d'une classe

- On appelle **membres** d'une classe, les attributs et méthodes qu'elle contient.
- Une classe peut contenir les membres suivants:

Attributs	
	Variables
	Constantes
Méthodes	
	Constructeur
	Destructeur
	Méthodes simples
	Propriétés
	Evénements
	Indexeurs
	Opérateurs

Règles de base pour une classe en bonne et due forme

- Une classe doit être enregistrée dans un fichier portant l'extension ***.cs***
  Le nom du fichier importe peu mais doit tout de même permettre d'identifier son contenu.
- Une classe se déclare toujours via le mot clé : **class**
- Le corps d'une classe commence à la première **{** et se termine à la dernière **}**
- Les attributs d'une classes doivent être déclarés en dehors de toute méthode
- Une classe peut contenir autant de méthodes que nécessaire
- Un fichier ***.cs*** peut contenir de 1 à n classes

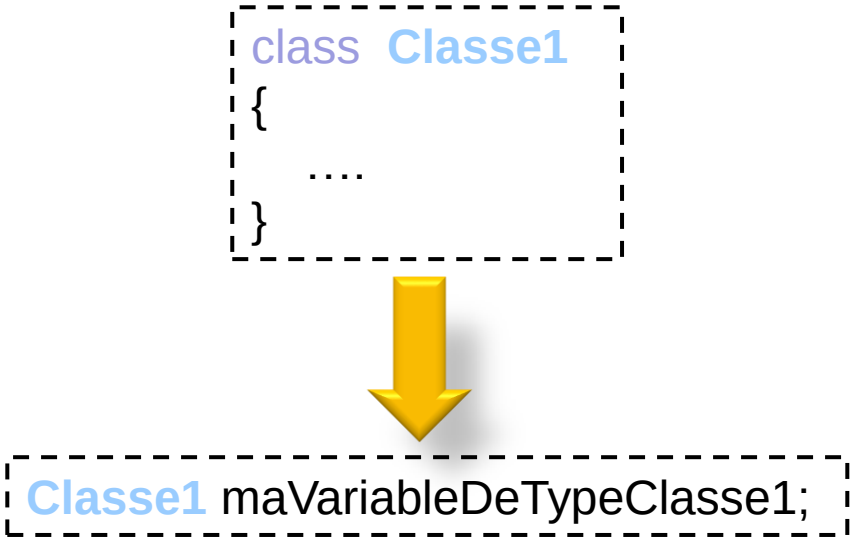
Notre premier objet sous forme de classe

- Il est temps de créer notre première classe

- Reprenez l'objet que vous aviez défini au chapitre précédent et matérialisez-le sous forme de classe, définissez le patron de votre objet en spécifiant ses attributs et méthodes.

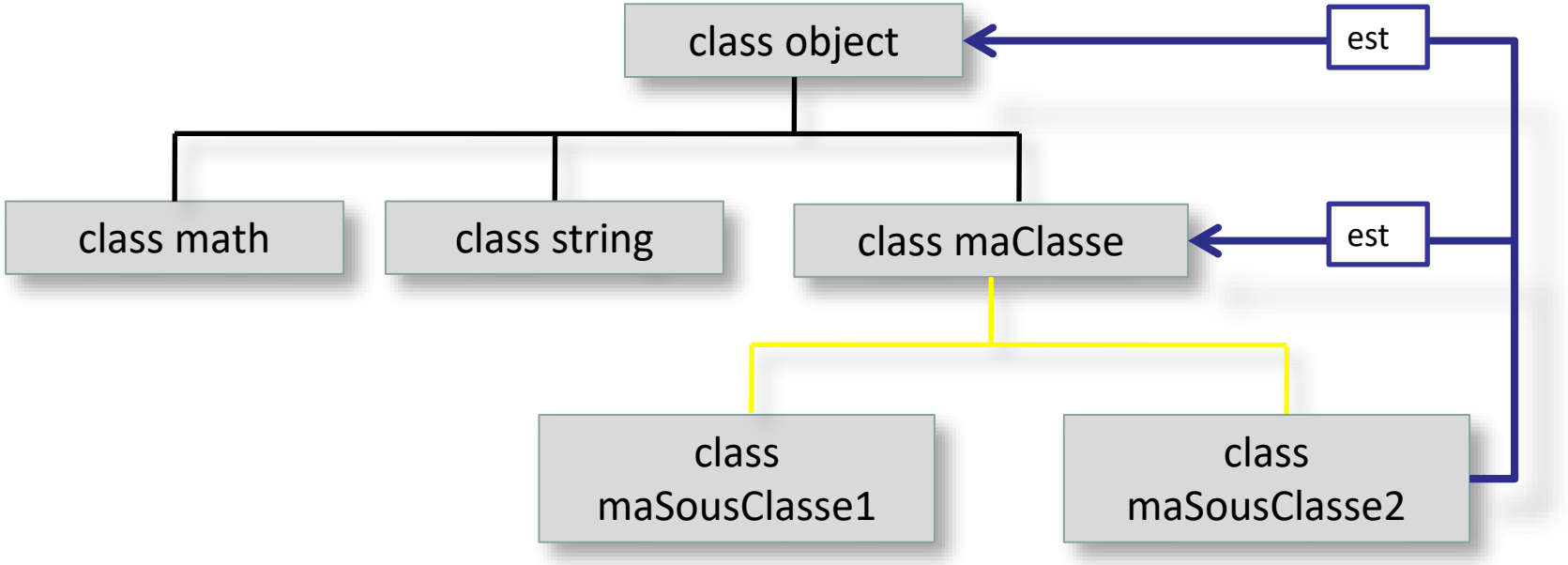
Une classe est un type en C#

- Une classe C# peut être considérée comme un nouveau **type** dans le programme et donc des variables peuvent être déclarées selon ce nouveau type.



Toutes les classes ont le même ancêtre

- En C#, toute classe hérite, indirectement ou pas, de la classe **object**
- Lorsque l'on ne précise pas de classe héritée dans la déclaration de la classe, celle-ci hérite implicitement de la classe object



La classe object

- Il s'agit de la classe de base fondamentale parmi toutes les classes en C#. Elle constitue la racine de la hiérarchie des types.

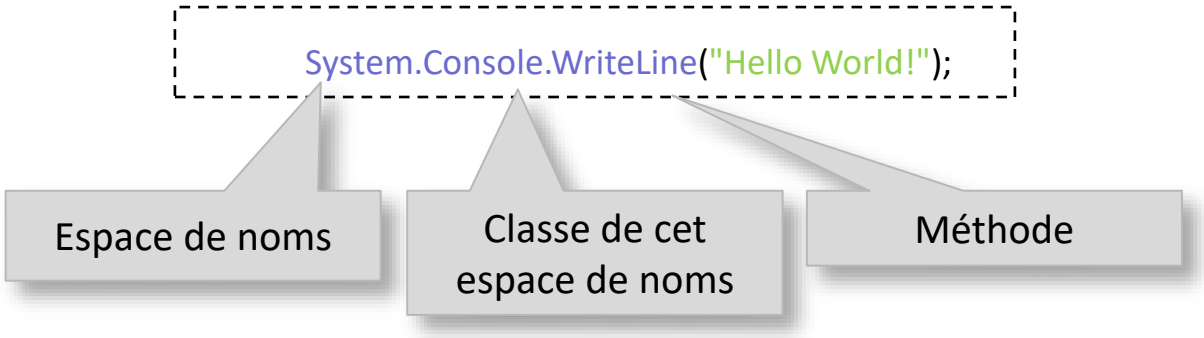
- Elle possède 4 méthodes dont hériteront toutes les autres classes:

  - ***GetType()*** *:* retourne le type de l'objet
  - ***Equals(object o)*** : permet de définir comment 2 objets sont égaux
  - ***GetHashCode()*** : retourne le code hash de l'objet (son n° de série en quelque sorte)
  - ***ToString()*** : retourne le type de l'objet sous forme d'une chaîne de caractères

- L'utilité de ses méthodes, et surtout le fait de les redéfinir dans d'autres classes, sera vu plus loin.

La notion de namespace

- Les **espaces de noms** ou namespaces sont énormément employés en programmation C#, de deux manières.
- Premièrement, le Framework .Net utilise des espaces de noms pour organiser ses nombreuses classes :



Le mot clé **using** peut être utilisé pour que le nom entier ne soit pas requis :

```
using System;
…
Console.WriteLine("Hello World!");
```

La notion de namespace

- Deuxièmement, déclarer ses propres espaces de noms peut vous aider à contrôler la portée des noms de classes et de méthodes dans les projets de programmation plus volumineux. Utilisez le mot clé **namespace** pour déclarer un espace de noms, comme dans l'exemple suivant :

```
using System;

namespace SampleNamespace
{
        class SampleClass
        {
                public void SampleMethod()
                {
                        Console.WriteLine("Hello World!");
                }
        }
}
```

La notion de namespace

- En résumé, les espaces de noms possèdent les propriétés suivantes :
  - ils organisent les grands projets de code
  - ils sont délimités avec l'opérateur .
  - le mot clé **using** signifie que vous n'avez pas besoin de spécifier le nom de l'espace de noms pour chaque classe
  - que vous déclariez ou pas explicitement un espace de noms dans un fichier source C#, le compilateur ajoute un espace de noms par défaut. Cet espace de noms sans nom, parfois appelé espace de noms global, est présent dans chaque fichier. Toute classe dans l'espace de noms global est disponible pour être utilisé dans un espace de noms nommé.
  - En C#, utiliser un namespace ne donne pas accès aux namespaces contenant ce dernier ni à ceux contenus. Il n'existe pas de syntaxe **using unnamespace.*;**.
    Vous devez préciser chaque namespace utilisé.

La notion de namespace - Exercice

- Dans un projet vide, créer :

  - 3 namepaces :
    A , AA (contenu dans A) et **B**

  - 3 classes :
    K dans le namespace **AA , L** dans le namespace **A , M** dans le namespace **B**

  - 1 classe **:**
    avec une méthode main dans le namespace **global** qui crée 3 objets respectivement de type **K**, **L** et **M**

La notion d’assembly

- Terme important en C# et dans tout le monde .Net, l'**assembly** désigne un ensemble de classes de n'importe quel langage .NET (VB.NET, C#) et de ressources (images, textes, videos).
- Si un projet comporte n classes qui compilées forment un exécutable, ces n classes forment ce que l’on appelle un(e) assembly.
- Un(e) assembly peut se présenter sous la forme d’une application exécutable (console ou windows) ou d’un fichier dll.
- Pour pouvoir utiliser les classes et ressources d’un(e)Assembly dans une autre, il faut référencer celle-ci.

Modificateurs d’accès

- Une classe C# peut se voir attribuer un modificateur de comportement sous la forme d'un mot clé devant la déclaration de classe.
- Par défaut, si aucun mot clef n'est indiqué, la classe est visible dans tout le namespace dans lequel elle est définie.
- Il y a 4 qualificateurs possibles pour modifier le comportement de visibilité d'une classe :

  public, **private**, **protected**, **internal** et la combinaison **protected internal**

- Sans qualificateur d’accès, une classe C# est considérée comme internal**.**

Modificateurs d’accès

Modificateurs d’accès	Explication
public class maClasse {…}	La classe est accessible à partir de n'importe quel endroit de n'importe quelle application
protected class maClasse {…}	La classe n'est accessible qu'à partir de ces classes enfants
internal class maClasse {…}	La classe n'est accessible que dans l'assembly où elle est définie
private class maClasse {…}	La classe n'est accessible qu’à partir de la classe qui la contient .
protected internal class maClasse {…}	La classe n'est accessible qu'à partir de ses classes enfants **OU** dans le même assembly
class maClasse {…}	qualifiée d’internal

Nous verrons, dans le chapitre, suivant comment ces modificateurs peuvent être appliqués aux membres d’une classe et ce que cela implique sur ceux-ci.

Modificateurs d'accès