

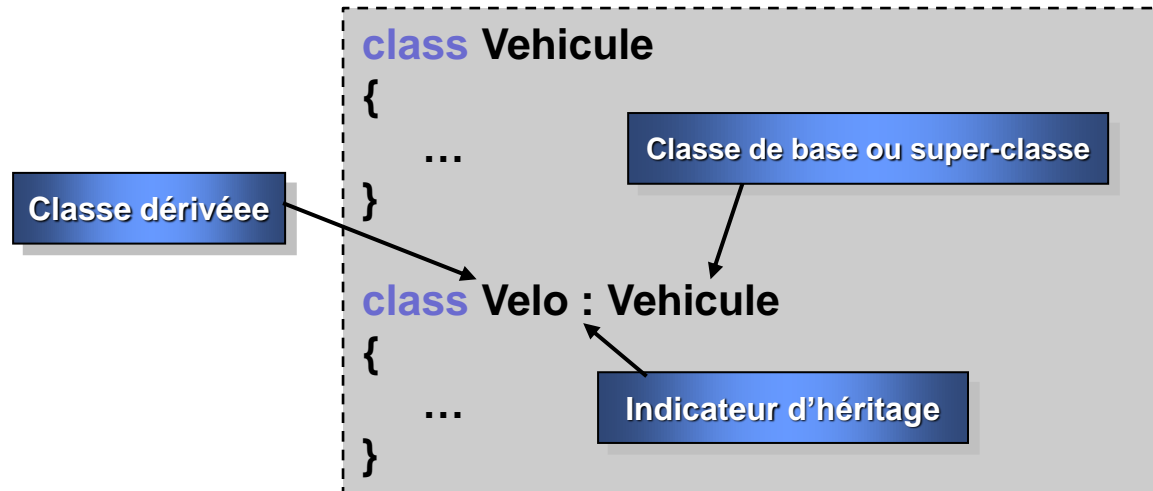
# Orienté Objet : L'héritage en C#

# L'héritage en C#

- L'héritage va, dans un premier temps, nous apporter la capacité à factoriser notre code et surtout à spécialiser nos classes. En orienté objet, le proverbe « mettre tous ses œufs dans le même panier » est à proscrire.
- Les membres définis dans une classe (pour autant que leurs visibilités leur permettent) seront communs et utilisables par toutes les classes qui hériteront directement ou indirectement de celles-ci.

# Etendre une classe à partir d'une autre

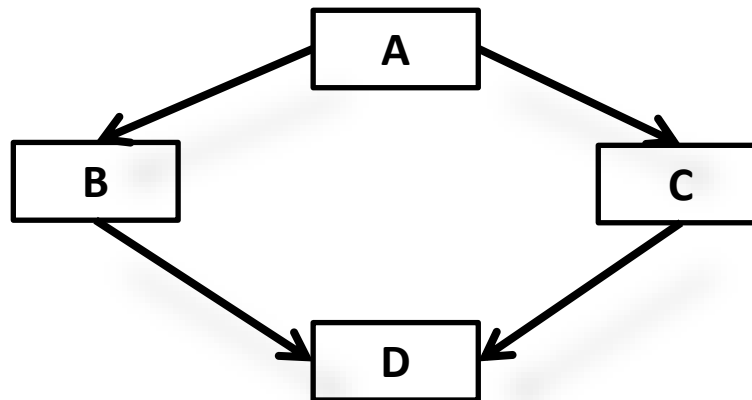
- Pour indiquer qu'une classe étend une autre classe, il faut le faire dans la déclaration de la classe qui va étendre, en utilisant les deux points (:) suivi du nom de la classe étendue.



- Une classe dérivée ne peut jamais avoir un niveau d'accès supérieur à celui de la classe qu'elle étend.
- Une classe ne peut étendre qu'une et une seule autre classe.


# Pourquoi pas de multi-héritage en C# ?


- Certains langages autorisent une classe à hériter de plusieurs classes...
- Un autre cas possible est ce que l'on appelle l'effet diamant



# Accéder aux membres d'une classe

```
class Token
{
    ...
    protected string name;
}
class CommentToken: Token
{
    ...
    public string Name( )
    {
        return name;
    }
}

class Outside
{
    CommentToken t = CommentToken();
    t.name 
}
```



# Appeler les constructeurs et méthodes d'une super-classe – le mot clé base

- Le mot clé base sert à accéder aux membres de la classe de base à partir d'une classe dérivée :
  - Appelle une méthode de la classe de base qui a été substituée par une autre méthode.
  - Spécifie quel constructeur de classe de base devrait être appelé lors de la création d'instances de la classe dérivée.
- Un constructeur déclaré “private” dans une classe de base ne peut jamais être appelé dans une classe dérivée.
- Le fait d'utiliser le mot clé base à partir d'une méthode statique constitue une erreur.

# Appeler les constructeurs et méthodes d'une super-classe – le mot clé base

- Si l'on désire qu'un constructeur d'une classe dérivée fasse appel à un constructeur d'une classe mère alors il faut utiliser la syntaxe suivante :

```
public MonConstructeur(paramètre1, paramètre2,...) : base([paramètre1, paramètre2,...])
```

*L'utilisation de paramètres va permettre de préciser à quel constructeur de la classe de base il faut faire appel.*

- Pour invoquer une méthode de la super classe, il faut utiliser cette syntaxe :

```
base.nomDeLaMéthode([paramètre1, paramètre2,...])
```

# Méthodes virtuelles et override de méthode

- Une des fonctionnalités intéressantes de la POO, et plus particulièrement de l'héritage, est le fait de pouvoir réaliser un override des méthodes d'une super-classe.

Si dans la classe de base, nous avons une méthode:

```
public virtual methodeAOverride() {...}
```

alors la syntaxe suivante doit être employée pour faire un override de la méthode:

```
public override methodeAOverride() {...}
```

- Les méthodes déclarées virtual sont considérées comme des méthodes polymorphiques et ne peuvent jamais être déclarées static et ou private
- Une méthode override ne peut jamais être déclarée static et ou private
- La méthode virtuelle et la méthode qui override doivent avoir la même signature



## Où sont les bugs ?


```
class Base
{
    public void Alpha( ) { ... }
    public virtual void Beta( ) { ... }
    public virtual void Gamma(int i) { ... }
    public virtual void Delta( ) { ... }
    private virtual void Epsilon( ) { ... }
}

class Derived: Base
{
    public override void Alpha( ) { ... }
    protected override void Beta( ) { ... }
    public override void Gamma(double d) { ... }
    public override int Delta( ) { ... }
}
```

# Les classes qui ne peuvent pas être étendues

- Une classe marquée comme **sealed** ne peut pas être étendue

```
public sealed class String
{
    ...
}

public class FancyString: String 
{
    ...
}
```

# Les méthodes qui ne peuvent pas être override

- Une méthode marquée comme **sealed** ne peut pas être réécrite

```
public class A
{
    public virtual Coucou() {...}
}

public class B : A
{
    public sealed override Coucou() {...}
}

public class C : B
{
    public override Coucou() {...} X
}
```

# Les mots clés base et override - Exercice 1

- Créer une classe Person :

Person
- nom - prenom
- getInfos()

- Créer une classe Employee qui étend Person :

Employee
- id
- getInfos()

- Instancier un objet de la classe Employee et en afficher toutes les informations via la méthode `getInfos()` déclarée dans la classe Employee

## Les mots clés base et override - Exercice 2

- Créer une classe BaseClass et une classe DerivedClass qui hérite de BaseClass.
- Déclarer deux constructeurs dans la classe BaseClass : l'un avec un paramètre (un entier) et l'autre sans.
- Le constructeur avec paramètre initialise une variable de classe.
- Instancier deux objets de la classe dérivée de manière à utiliser les constructeurs de la super-classe.