Héritage

En C++ comme dans tous les langages orientés objet, une classe peut **hériter** d'une autre.

Elle possède alors les mêmes attributs et méthodes, en plus d'avoir les siennes propres.

Signification : le verbe "être"

On dit qu'une classe Bidule peut hériter d'une classe Truc si d'un point de vue des concepts représentés on peut dire qu'un Bidule **est** un Truc. Bidule est alors la classe fille, Truc la classe mère.

Exemple : Un *Velo* **est** un *Vehicule*, donc on peut faire en sorte que la classe *Velo* hérite de la classe *Vehicule*. *Velo* est donc la classe fille et *Vehicule* la classe mère. *Velo* pourra donc hériter des différentes méthodes et attributs de la classe *Vehicule*. On peut tout simplement et pour se simplifier la vie considérer que tout objet de classe *Velo* **est** un *Vehicule*.

Syntaxe C++

```
class Velo : public Vehicule{
// contenu spécifique de la classe Velo (définition des nouvelles méthodes & attributs)
};
```

Autre exemple :
Un Mage est un Personnage, mais un Personnage n'est pas forcément un Mage.
Mage peut donc hériter de la classe Personnage.
Un Mage est juste un type de Personnage, davantage spécifique.

```
class Mage : public Perso{
// contenu spécifique de la classe Mage (définition des nouvelles méthodes & attributs)
};
```

Il faudra évidemment aussi, en plus du fichier .h de déclaration des attributs et méthodes, un .cpp pour les implémentations de nouvelles méthodes. Exactement comme d'habitude.

Intérêt

L'intérêt de l'héritage est qu'**un pointeur vers un objet d'une classe fille** peut toujours être utilisé à la place d'**un pointeur vers un objet de sa classe mère**.

Ainsi, si un Velo est un Vehicule alors on peut utiliser un vélo à chaque fois qu'on aurait eu besoin d'un véhicule. L'inverse n'est, en revanche, pas vrai.

Si on sait réparer n'importe quel véhicule on sait forcément réparer un vélo. En revanche savoir réparer un vélo ne signifie pas savoir réparer n'importe quel véhicule.

Exemple avec la classe Mage, qui hérite de la classe Perso, mais en y ajoutant des points de magie. Le constructeur devient, implémenté dans le .cpp :

Mage::Mage() : Personnage(), _magie(100) {}

Public, private… et protected

On avait vu public (accessible partout) et private (accessible seulement depuis la classe). Il existe aussi le mot-clé protected, qui se comporte comme private, mais en permettant également l'accès depuis les classes filles.

Ainsi, les méthodes d'un Mage ont également accès aux attributs protected hérités du Perso, pas seulement aux nouveaux attributs définis dans la classe Mage. En revanche, elles n'ont pas accès aux attributs private hérités du Perso. Seules les méthodes de la classe Perso y ont accès !

Remplacer une méthode

Une classe fille peut remplacer l'implémentation des méthodes de sa classe mère.

Les méthodes de la classe fille remplacent celles de la classe mère quand elles existaient déjà (on peut y accéder quand même en faisant ClasseMere::nomDeLaMethode();)

Au moment de s'en servir, c'est le type de la variable qui détermine quelle méthode appeler, et non sa vraie nature.