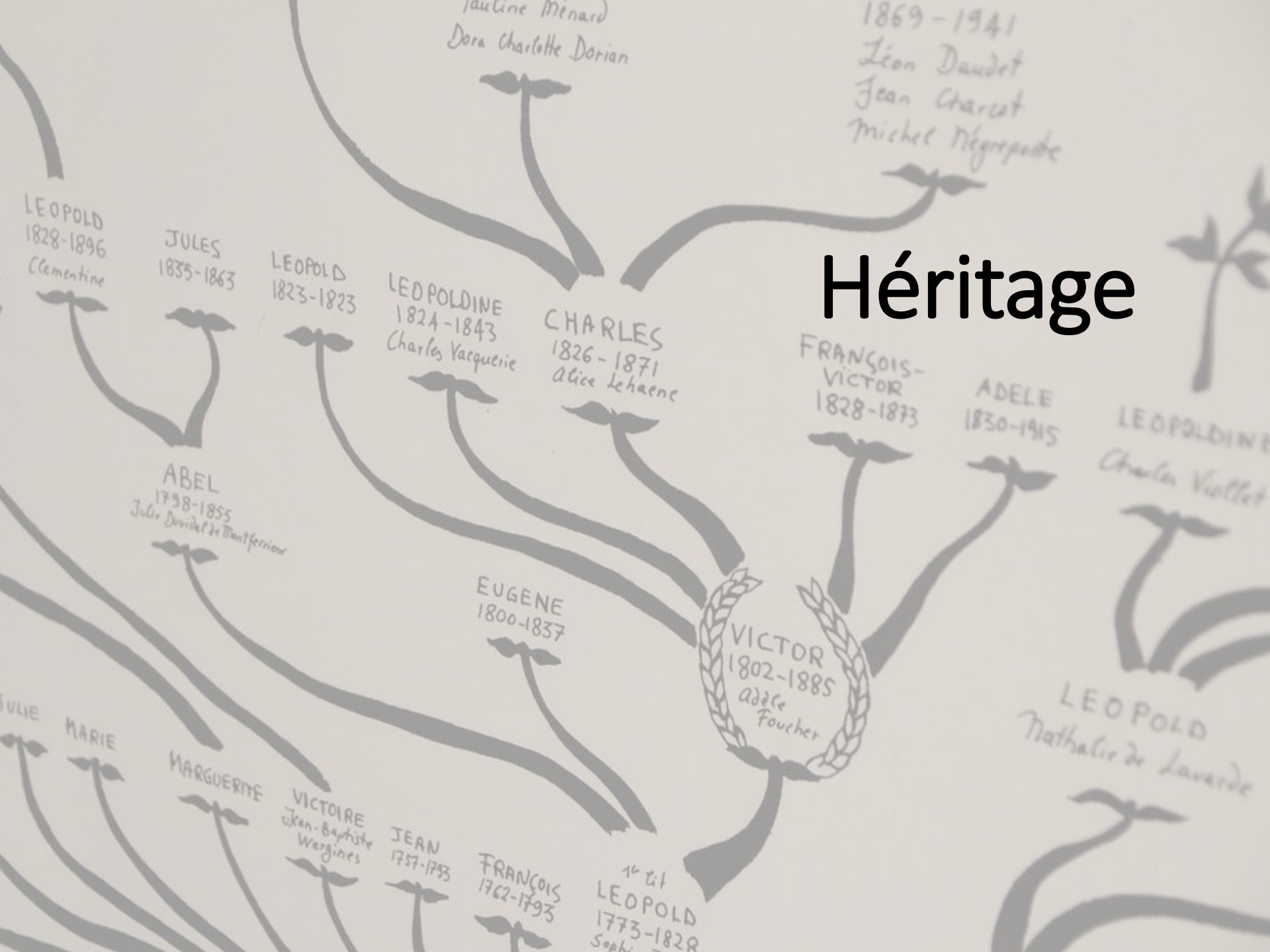

Pauline Ménard
Dora Charlotte Dorian
1869-1941
Léon Daudet
Jean Charcot
Michel Négrepaste
LEOPOLD
1828-1896
Clementine
JULES
1835-1863
LEOPOLD
1823-1823
LEOPOLDINE
1824-1843
Charles Vacquerie
CHARLES
1826-1871
Alice Lehaene
FRANÇOIS-VICTOR
1828-1873
ADELE
1830-1915
ABEL
1798-1855
Julie Duvidal de Montferrier
EUGENE
1800-1837
VICTOR
1802-1885
Adèle Foucher
MARIE
MARGUERITE
VICTOIRE
Jean-Baptiste Wargines
JEAN
1757-1793
FRANÇOIS
1762-1793
LEOPOLD
1773-1828
LEOPOLD
Nathalie de Lavarde

Héritage

Reprenons un projet…

```
class Point {
    private :
        int x, y;
    public :
        Point();
        Point(int, int);
        void afficher();
};
```

On souhaite ajouter une fonctionnalité permettant de créer des points colorés tout en conservant l'existant.

Comment faire ?

Jusque-là…

On recopiait la classe en ajoutant les nouvelles fonctionnalités :

```
class PointCouleur {
    private :
        int x, y;
        string couleur;
    public :
        PointCouleur(int, int, couleur);
        void afficher();
};
```

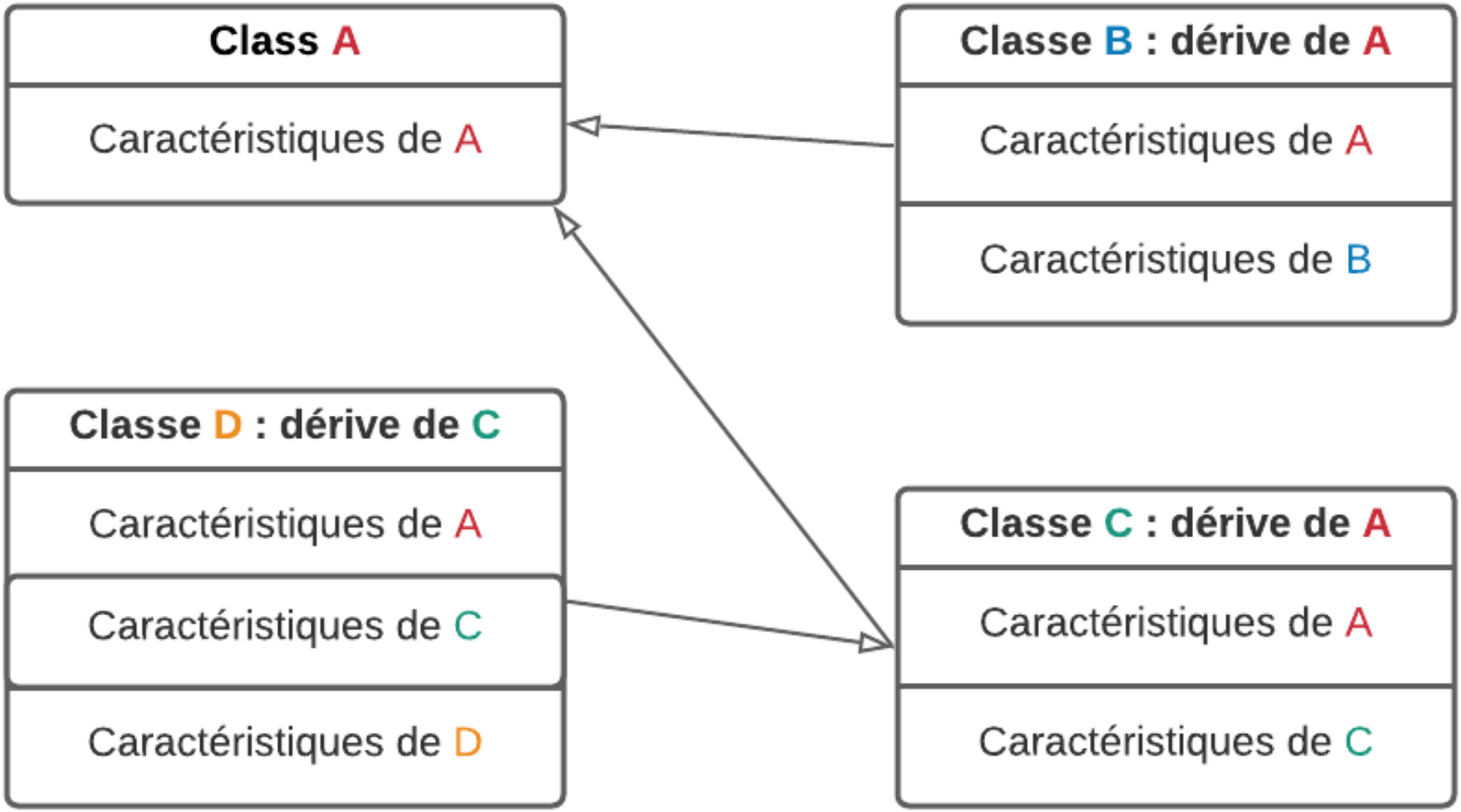
Oui… mais… si on souhaite faire **évoluer** la classe Point… On va devoir recopier tout le code d'une classe dans l'autre ?

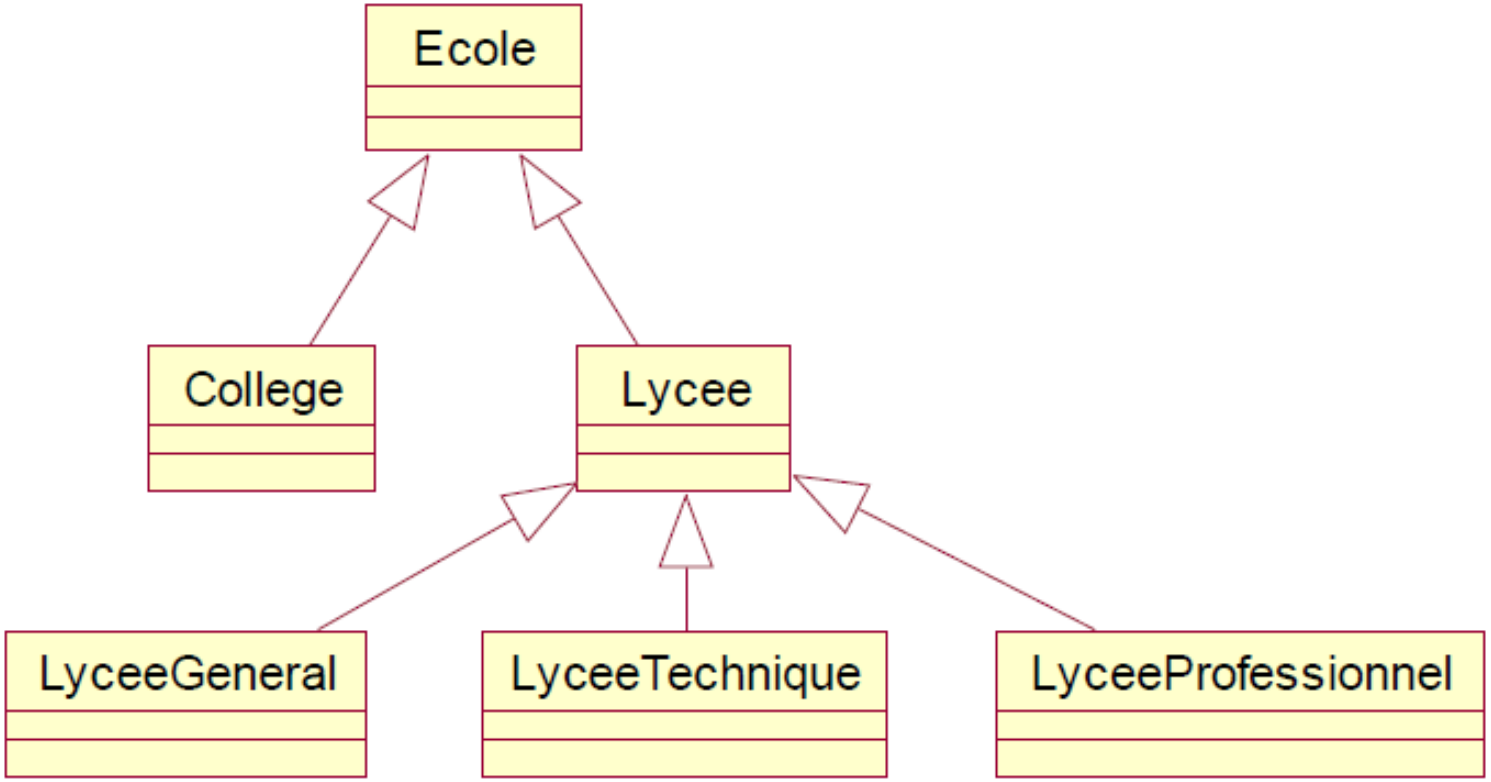
Un concept fondamental

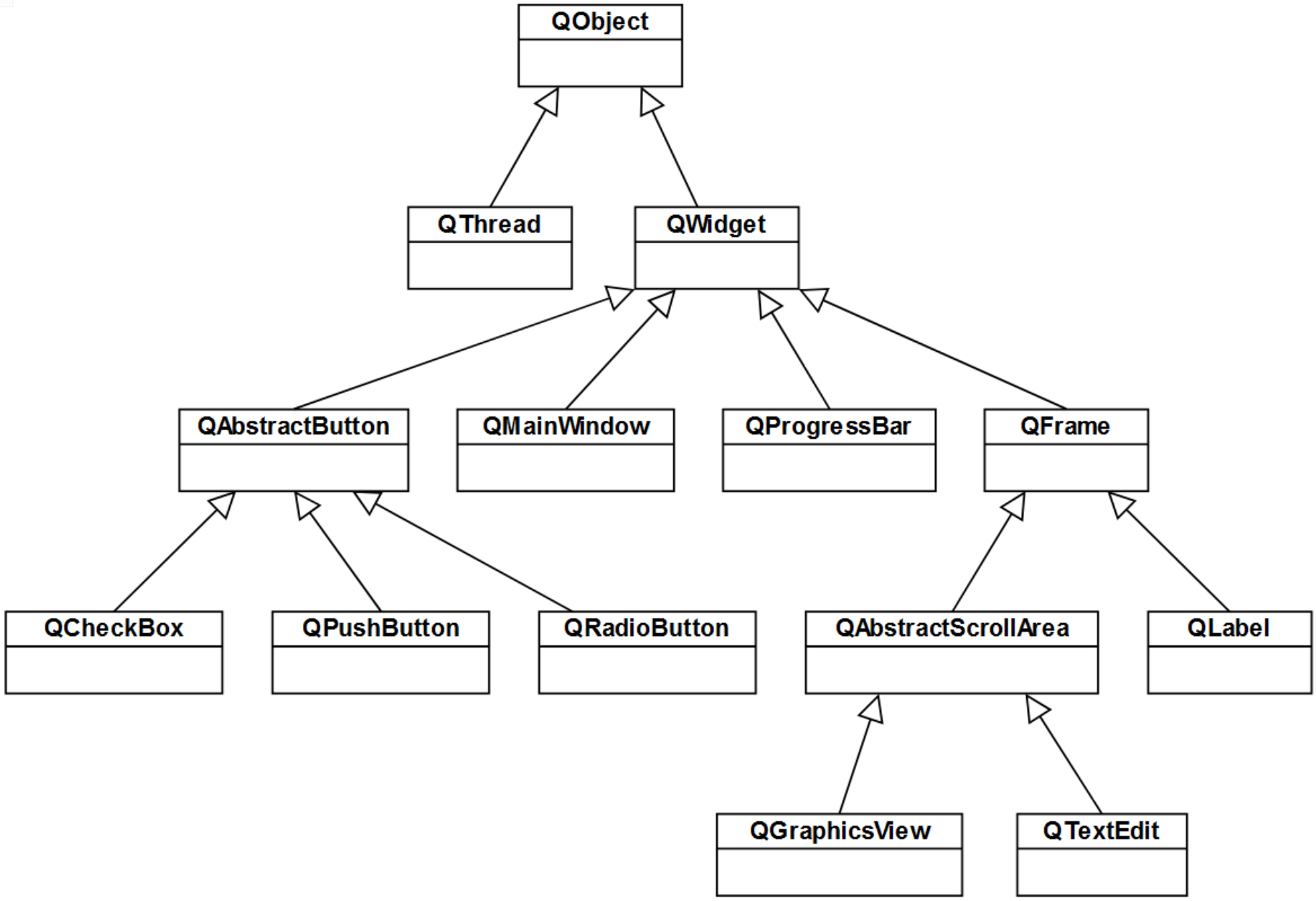
- Le concept d'**héritage** constitue l'un des fondements de la P.O.O.

- Il permet de définir une nouvelle classe dite dérivée, à partir d'une classe de base existante.

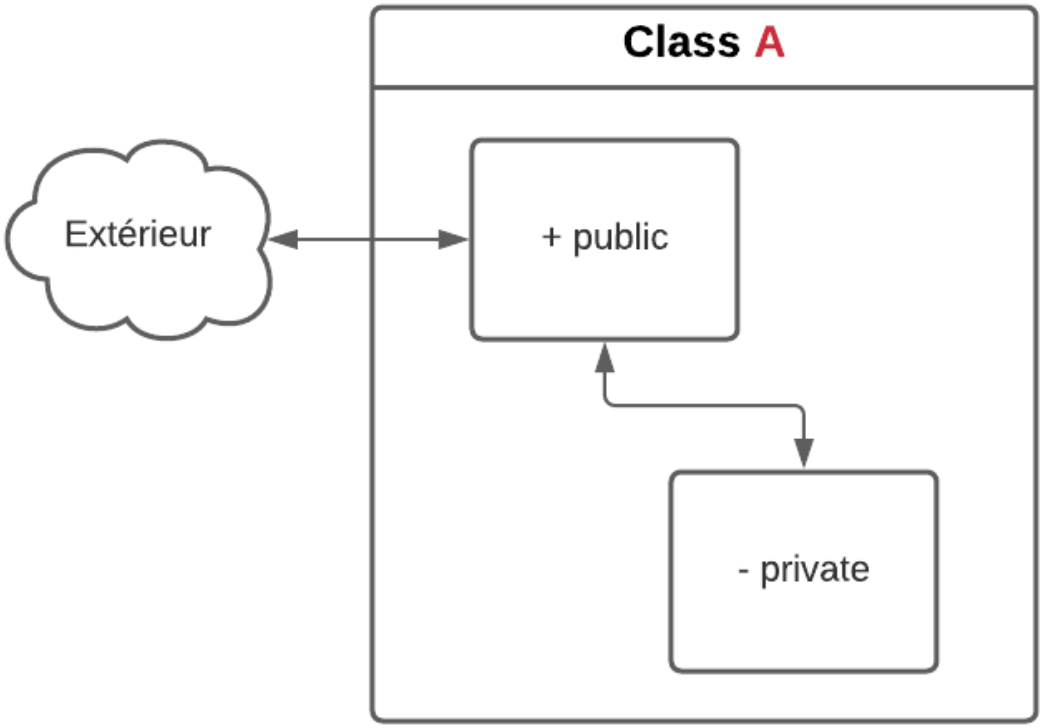
- La classe dérivée héritera des caractéristiques de la classe de base, tout en lui en ajoutant de nouvelles, et ceci sans modifier quoi que ce soit à la classe de base.


Class A
Caractéristiques de A
Classe B : dérive de A
Caractéristiques de A
Caractéristiques de B
Classe D : dérive de C
Caractéristiques de A
Caractéristiques de C
Caractéristiques de D
Classe C : dérive de A
Caractéristiques de A
Caractéristiques de C

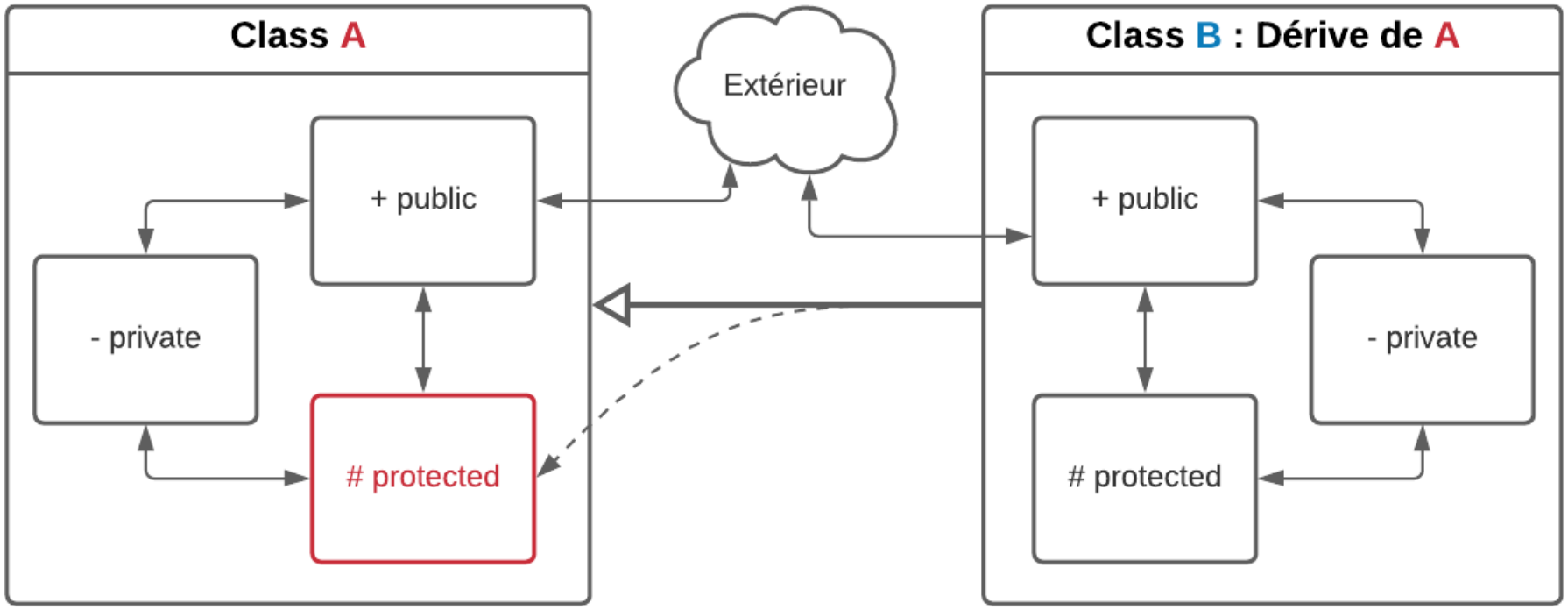

Ecole
College
Lycee
LyceeGeneral
LyceeTechnique
LyceeProfessionnel


QObject
QThread
QWidget
QAbstractButton
QMainWindow
QProgressBar
QFrame
QCheckBox
QPushButton
QRadioButton
QAbstractScrollArea
QLabel
QGraphicsView
QTextEdit

Spécificateurs d’accès (membres)



protected


Class A
+ public
- private
protected
Extérieur
Class B : Dérive de A
+ public
- private
protected

Déclaration

```
class B : public A {
    ...
};
```

Type de dérivation	Accès dans A	Accès dans B
public	public	
	private	
	protected	
private	public	
	private	
	protected	
protected	public	
	private	
	protected	

Constructeurs et destructeurs

```
class Point {};
class PointCouleur : public Point {};
int main() {
     PointCouleur a;
}
```

Construction

Instanciation d'un objet `PointCouleur`

↓

Appel du constructeur de `Point`

↓

Appel du constructeur de `PointCouleur`

Destruction

Destruction d'un objet `PointCouleur`

↓

Appel du destructeur de `PointCouleur`

↓

Appel du destructeur de `Point`

Accès aux attributs de la classe mère

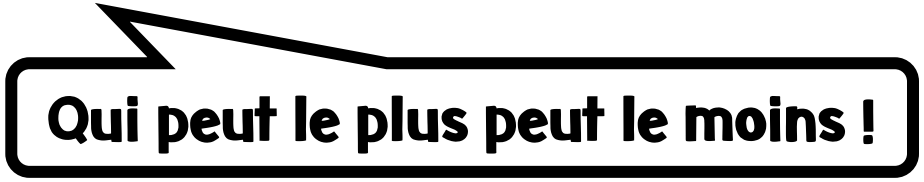
```
class Point {
    protected :
        int x, y;
    public :
        Point();
        Point(int, int);
        void afficher;
};

class PointCouleur : public Point {
    private :
        string couleur;
    public :
        PointCouleur();
        PointCouleur(int _x, int _y, string _couleur) {
            x = _x;
            Point::y = _y;
            couleur = _couleur;
        };
};
```

Passage des arguments entre constructeurs

```
class Point {
   private :
      int x, y;
   public :
      Point(int _x, int _y) {
          x = _x;
          y = _y;
      };
};
class PointCouleur : public Point {
   private :
      string couleur;
   public :
      PointCouleur(int _x, int _y, string _couleur) : Point(_x, _y) {
          couleur = _couleur;
      };
};
```

Compatibilité entre classe de base et classe dérivée

D'une manière général en POO on considère qu'un objet d'une classe dérivée peut remplacer un objet de classe de base.




```
Point a, *c;
PointCouleur b, *d;

// Affectations autorisées
a = b;
c = d;

// Affectation interdites
b = a;
d = c;
```

Surdéfinition des fonctions membres

```
class Point {
  public :
    void afficher(){
      cout << "(" << x << "," << y << ")";
    };
};
class PointCouleur : public Point {
  public :
    void afficher() {
      Point::afficher();
      cout << " de couleur " << couleur;
    };
};
```

La surdéfinition d'une méthode cache toutes les surdéfinitions de la classe de base (et des ascendantes).

La recherche d'une fonction s'arrête à un niveau.

```
class A {
    public :
        void f(int n) { ... };
        void f(char c) { ... };
};
class B : public A {
    public :
        void f(int, int) { ... };
};
int main() {
    int n; char c ; B b ;
    b.f(n);         // erreur de compilation
    b.f(c);         // erreur de compilation
    b.f(n, n);      // OK
}
```

Le constructeur de recopie

```
class A {...};
class B : public A {...};

void fct(B);

int main() {
    B b1;
    fct(b1); // Passage par valeur donc recopie de l'objet
}
```

Si la classe dérivée (B) ne définit pas de constructeur de recopie :

- Il y a appel du constructeur de recopie par défaut de B pour les données membre de B. Pour les données de b1 appartenant à la classe A on va chercher à appeler le constructeur de recopie de A.
- Si A a défini un constructeur de recopie, il sera appelé, sinon on fera appel au constructeur de recopie par défaut.

Le constructeur de recopie

```
class A {...};
class B : public A {...};

void fct(B);

int main() {
    B b1;
    fct(b1); // Passage par valeur donc recopie de l'objet
}
```

Si la classe dérivée définit un constructeur de recopie

- Le constructeur de B est appelé mais aucun appel au constructeur (défaut ou non) de recopie de A.
- Le constructeur de recopie de la classe dérivée doit prendre en charge l'intégralité de la recopie de l'objet, partie héritée incluse.
- Il reste possible d'utiliser le mécanisme de transmission d'informations entre constructeurs :

```
B (B &x) : A(x) {...}
```

L'opérateur d'affectation

Si la classe dérivée ne surdéfinit pas l'opérateur =

- Les membres propres à B sont traités par l'affectation par défaut. La partie héritée de A est traitée par l'affectation prévue dans la classe A soit:
  - Par l'opérateur = surdéfini dans A
  - Par l'affectation par défaut

Si La classe dérivée surdéfinit l'opérateur =

- L'affectation fera appel à l'opérateur = défini dans B. Celui de A ne sera pas appelé.
- Il faudra que l'opérateur = de B prenne en charge l'affectation de sa partie et de la partie héritée