Centrale-Lyon - Programmation Orientée Objet - Cours Info 2A (1/4)

Introduction à la Programmation Orientée Objet et au C++

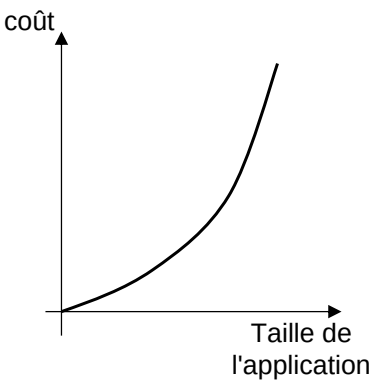
- Pourquoi la Programmation par objets
- UML
- L'approche objet
- Les langages à objets
- Le C++, principales notions
- Deux exemples

Version 2.2 – 8/11/2005

Pourquoi la Programmation Orientée Objet ?

Problème de crise du logiciel:

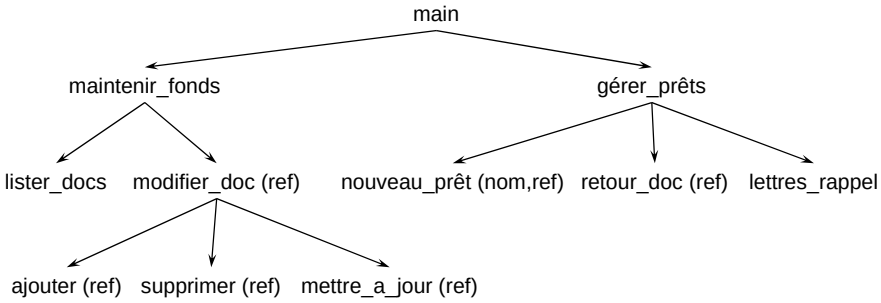
- Le matériel est de moins en moins cher
- les applications sont de plus en plus complexes:
  - complexité des problèmes à résoudre
  - complexité du processus de développement
  - flexibilité croissante des logiciels
- la durée de vie du logiciel doit être augmentée
- Le coût croit exponentiellement avec la complexité



Cours Info 2A (1/4) Introduction à la POO et au C++ 2

Limites de l'approche fonctionnelle (1/2)

- Découpage fonctionnel d'un problème informatique :
  - une approche intuitive
  - exemple : gestion d'une bibliothèque



  - hiérarchie de fonctions réalisant les services désirés

Cours Info 2A (1/4) Introduction à la POO et au C++ 3

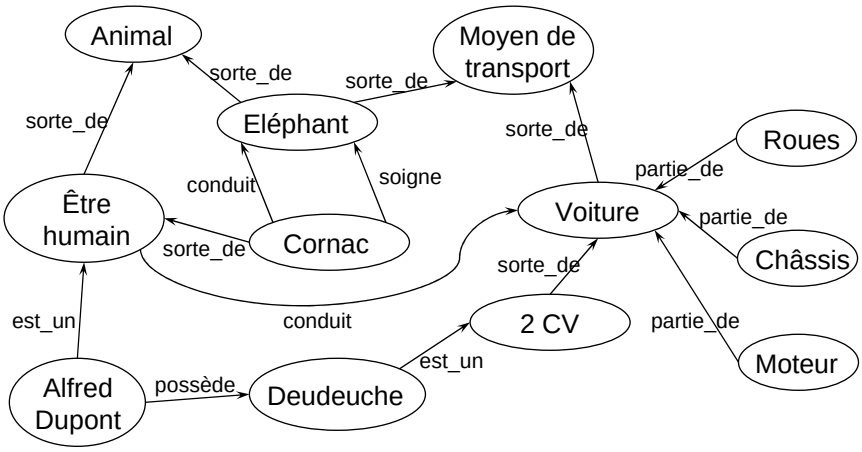
Limites de l'approche fonctionnelle (2/2)

- Avantage de l'approche fonctionnelle :
  on peut réutiliser des fonctions génériques déjà définies pour créer de nouvelles fonctions :
  - exemple : la fonction "mettre_a_jour (ref)" peut être utilisée par "nouveau_prêt" et "retour_doc"
- MAIS : l'évolution du logiciel est délicate car les fonctions deviennent inter-dépendantes
  - modifier "mettre_a_jour" entraîne en cascade des modifications dans "modifier_doc", "nouveau_prêt" et "retour_doc"...
- Autre problème :
  - les données et les fonctions travaillant sur ces données sont dispersées
- Solution : regrouper les données et le code travaillant sur ces données dans une même entité ==> un OBJET

Cours Info 2A (1/4) Introduction à la POO et au C++ 4

Objectifs de la programmation par objets (1/3)

- Modélisation du monde réel :
  - Exemple : les réseaux sémantiques



Cours Info 2A (1/4) Introduction à la POO et au C++ 5

Objectifs de la programmation par objets (2/3)

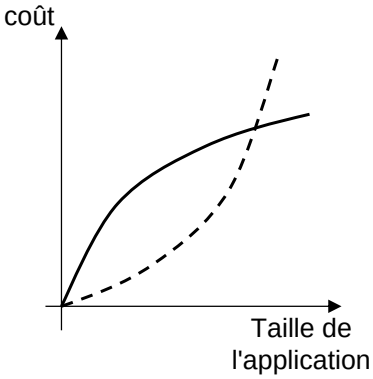
- Exploitation de la redondance :
  - dans le monde réel on a de nombreux représentants mais peu de concepts différents :
    - ➔ par exemple, il y a 6 milliards d'êtres humains sur terre mais une seule espèce : "homo sapiens sapiens"
- Réutilisation de composants logiciels :
  - utilisation de composants préexistants (objets) comme :
    - ➔ en électronique : composants, circuits intégrés, modules, etc…
    - ➔ en construction automobile : moteurs, roues, etc...
- Réduction de l'impact des modifications et extensions :
  - grâce au confinement dans des petites unités (objets)

Cours Info 2A (1/4) Introduction à la POO et au C++ 6

Objectifs de la programmation par objets (3/3)

- Réduction et explicitation du couplage entre différentes parties d'une application :
  - définition d'une interface minimale et clairement définie entre objets
- Protection et abstraction des entités :
  - en séparant l'interface de l'implémentation

==> Inverser la courbe :



Cours Info 2A (1/4) Introduction à la POO et au C++ 7

UML (Unified Modeling Language)

- Langage de modélisation objet né de la fusion de 3 méthodes de COO du début des années 90 : OMT, Booch et OOSE
- Normalisé en 1997 par l'OMG (Object Management Group) qui est aussi à l'origine de CORBA
- UML:
  - un langage graphique (supporté par un méta-modèle)
  - un ensemble de vues (=diagrammes) modélisant les aspects statiques et dynamiques
- Il ne propose pas UNE méthodologie d'analyse et de conception, mais peut les supporter toutes !

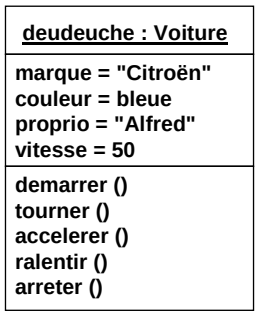
⇨ Dans le cadre de ce cours, nous n'utiliserons qu'un sous-ensemble **très réduit** de UML pour illustrer les principes de la POO

Nous n'utiliserons que les diagrammes de classes et d'objets

Cours Info 2A (1/4) Introduction à la POO et au C++ 8

Objet : définition

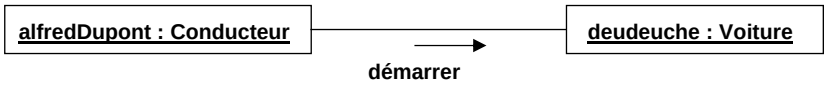
- Un objet est une entité qui possède :
  - une **identité** : nom qui permet de distinguer un objet d'un autre objet
  - un **état** : ensemble de valeurs associées à des propriétés et qui caractérisent l'objet à un instant t
  - un **comportement** : ensemble de services ou d'opérations que peut rendre un objet ou qui modifient son état



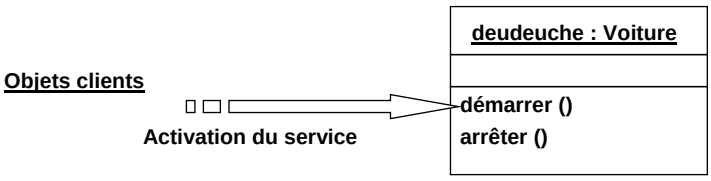
Cours Info 2A (1/4) Introduction à la POO et au C++ 9

Communication entre objets

- Les objets communiquent entre eux par l'envoi de messages
- Un message entraîne l'activation d'un service de l'objet



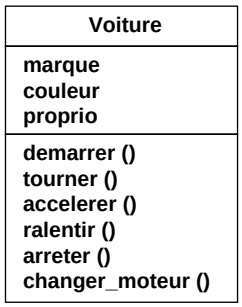
- message =
  - ➔ un sélecteur : identifie quel est le service de l'objet activé
  - ➔ des arguments : paramètres effectifs communiqués au service



Cours Info 2A (1/4) Introduction à la POO et au C++ 10

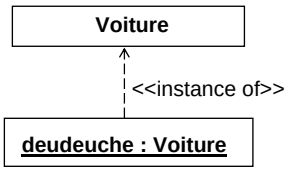
Notion de classe

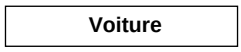
- Classe :
  - **type de donnée abstrait**
  - Il regroupe des propriétés communes à des objets :
    - ➔ attributs (≈ variables)
    - ➔ méthodes (≈ fonctions)



Classe documentée

- Instances d'un classe :
  - ce sont les objets dérivés de cette classe





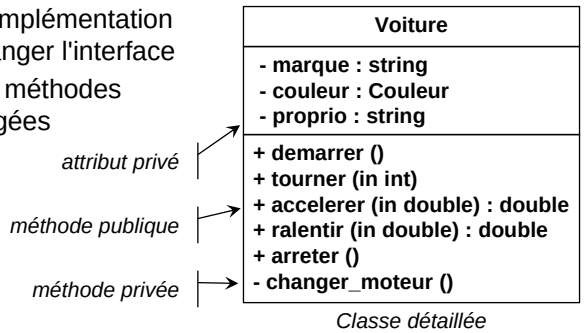
Classe non-documentée

- on peut faire les équivalences :
  - ➔ classe ≈ type
  - ➔ instance ≈ variable

Cours Info 2A (1/4) Introduction à la POO et au C++ 11

Concepts fondamentaux (1/3)

- Encapsulation :
  - masquer les détails de l'implémentation en définissant une interface
  - L'interface définit les services accessibles par les autres objets : méthodes publiques de l'objet
  - L'encapsulation facilite l'évolution car on peut changer l'implémentation de l'objet sans changer l'interface
  - Les attributs et les méthodes privées sont protégées (accessibles seulement par les méthodes de la classe)

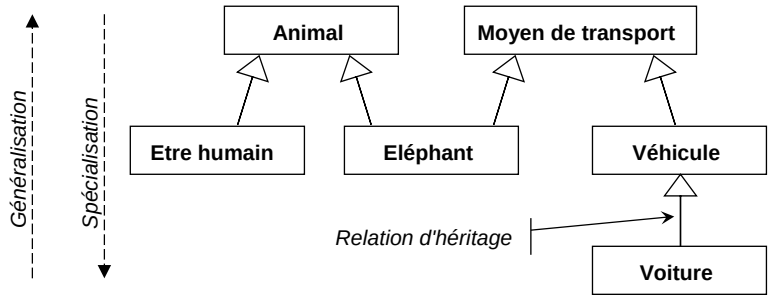


Classe détaillée

Cours Info 2A (1/4) Introduction à la POO et au C++ 12

Concepts fondamentaux (2/3)

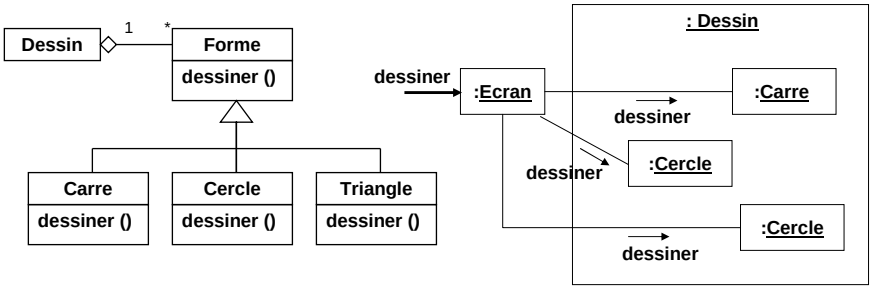
- Héritage (spécialisation et généralisation) :
  - Transmission des propriétés d'une classe vers une sous-classe
  - la spécialisation permet d'ajouter des caractéristiques ou d'en adapter certaines
  - la généralisation permet de factoriser des classes en regroupant des propriétés qui leur sont communes
  - l'héritage évite la duplication et favorise la réutilisation



Cours Info 2A (1/4) Introduction à la POO et au C++ 13

Concepts fondamentaux (3/3)

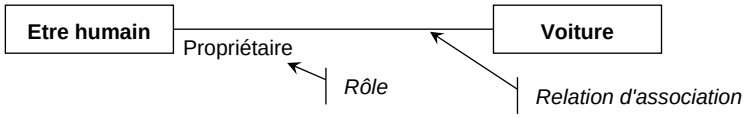
- Polymorphisme :
  - c'est un mécanisme qui permet à un objet client d'utiliser les services d'une interface « générique », sans savoir quel est l'objet qui va véritablement répondre à la requête
  - permet de créer des superclasses qui sont spécialisées par la suite



Cours Info 2A (1/4) Introduction à la POO et au C++ 14

Relations entre objets (1/2)

- Association :
  - relation signifiant que les instances de classes ont certaines liaisons entre elles (lien sémantique)

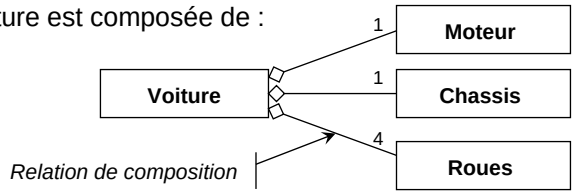


  - La cardinalité de la relation exprime le nombre d'instances pouvant être associées ; exemple :
    - ➔ un "être humain" peut ne pas avoir de voiture ou être propriétaire d'une ou plusieurs "voitures" (cardinalité 0..N)
    - ➔ une "voiture" est liée à un propriétaire et un seul (cardinalité 1)

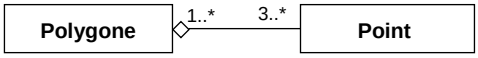
Cours Info 2A (1/4) Introduction à la POO et au C++ 15

Relations entre objets (2/2)

- Composition :
  - relation entre classes signifiant que les instances d'une classe sont les composants d'une autre classe
  - La composition permet d'assembler des objets pour en créer de plus complexes
  - Exemple : une voiture est composée de :
    - ➔ 1 moteur,
    - ➔ 1 châssis
    - ➔ et 4 roues



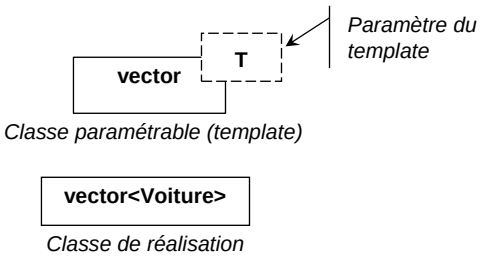
- Agrégation :
  - Composition faible : les objets agrégés ont une durée de vie indépendante de l'agrégat
  - Exemple :



Cours Info 2A (1/4) Introduction à la POO et au C++ 16

Autres concepts

- Classe abstraite :
  - classe très générale, non instanciable
  - utilisé pour la factorisation de propriétés dans des arbres d'héritage
- Généricité :
  - utilisation de classes paramétrables (templates en anglais, parfois appelées patrons) : classes génériques pouvant générer des familles de classes de réalisation concrètes (c-à-d instanciables)
  - exemple : classe paramétrable "vector <T>" permet de générer des :
    - → vecteurs d'entier
    - → vecteurs de réels
    - → vecteurs de Voiture
    - → etc...



Les langages objets (1/2)

- Emergence des langages objets:
  - Simula en 1967 :
    - → type de donnée abstrait, notion de classe
  - Smalltalk en 1976 :
    - → classes,
    - → associations, hiérarchies entre classes
    - → messages entre objets
  - C++ en 1980
- Langage objet :
  - met en œuvre tous les principes de la programmation objet
- Langage orienté objet :
  - met en œuvre quelques uns des principes de la programmation objet

Les langages objets (2/2)

- Langage objet natif :
  - Langages conçus autour du concept objet et qui ne peuvent manipuler autre chose
  - exemples : Smalltalk, Java
- Langage à extension objet :
  - langages traditionnels étendus pour manipuler des objets tout en conservant le fonctionnement traditionnel avec des variables
  - C --> C++, Objective C
  - Pascal --> Turbo Pascal objet
  - LISP --> Flavors, xlisp, CLOS
  - Prolog --> Emicat
  - Ada --> Ada 95

Cours Info 2A (1/4) Introduction à la POO et au C++ 19

Le langage C++

- Historique :
  - 1980 : création de C++ par Bjarne Stroustrup aux Bell labs (là où le C et Unix ont été créés)
  - 1986 : ouvrage de Stroustrup
    The C++ programming language
    définition par l'exemple du langage
  - 1990 : normalisation par l'ANSI de C++ dans sa version 2.0
  - 1998 : normalisation par l'ISO (norme ISO 14882)
- la langage C++ est une extension du langage C :
  - extensions objets
  - autres extensions corrigeant des insuffisances de C

Cours Info 2A (1/4) Introduction à la POO et au C++ 20

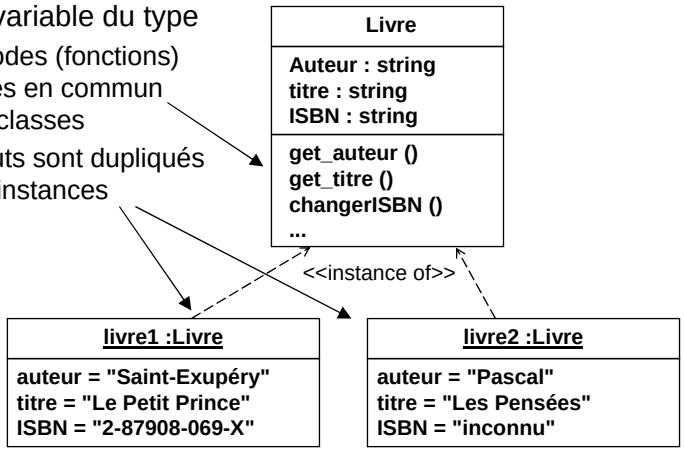
Les classes en C++

- Classe :
  - nouveau type défini par l'utilisateur : **class**
  - construction analogue à une structure (struct)
  - contient des déclarations d'attributs et des déclarations de méthodes (fonctions)
- Protection :
  - les données et fonctions situées dans la partie privée sont accessibles uniquement à travers les fonctions de la partie publique
  - partie privée introduite par **private:**
  - partie publique introduite par **public:**
  - Attention: le contenu d'une classe est privé par défaut
  - il existe aussi une partie protégée (introduite par **protected:**) similaire à la partie privée sauf pour l'héritage (notation UML : #)

Cours Info 2A (1/4) Introduction à la POO et au C++ 21

Classes et Instances

- Classe = nouveau type construit
- Instance = variable du type
  - les méthodes (fonctions) sont mises en commun dans les classes
  - les attributs sont dupliqués dans les instances



Cours Info 2A (1/4) Introduction à la POO et au C++ 22

Héritage en C++

- Héritage de classe : simple ou multiple
- Une classe dérivée est définie par rapport à une classe de base en introduisant 3 types de différences :
  - ajout d'attributs
  - ajout de fonctions membres
  - modifications de fonctions membres (par surcharge des méthodes)
- 3 modes d'héritage :
  - *private*, *protected* et *public* (voir poly)
- Héritage *public* (le plus utilisé) :
  - la zone *private* de la classe de base est **inacessible**
  - la zone *public* de la classe de base est accessible dans la classe dérivée et garde sont statut de zone *public*

Cours Info 2A (1/4) Introduction à la POO et au C++ 23

1er Exemple : classe Date (1/3)

- Création d'une classe Date :
  - 3 attributs entiers privés : jour, mois, an
  - 3 méthodes publiques :
    - → initialiser la date
    - → lire la date
    - → afficher la date
- Déclaration de la classe Date en C++ :

Date
- jour : int **- mois : int** **- an : int**
+ initialiser (in int, in int, in int) : bool **+ lire_date (out int, out int, out int)** **+ afficher_date ()**

```
class Date
{
  private:
    int jour, mois, an;
  public:
    bool initialiser (int, int, int);
    void lire_date (int &, int &, int &);
    void afficher () const;
};
```

Cours Info 2A (1/4) Introduction à la POO et au C++ 24

1er Exemple : classe Date (2/3)

- Définition des méthodes :

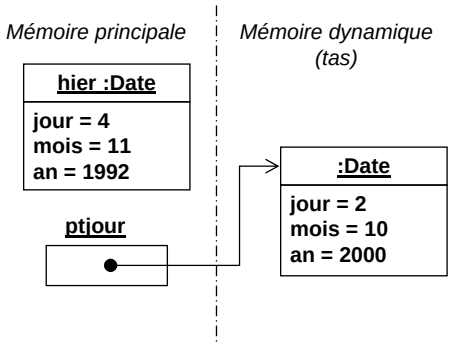
```
bool Date::initialiser (int j, int m, int a)
{
   if (j >= 1 && j <= 31)
      jour = j;
   else
      return false;
   if ( m >= 1 && m <= 12)
      mois = m;
   else
      return false;
   if (a > 0)
      an = a;
   else
      return false;
   return true;
}
```

```
void Date::lire_date (int& j, int& m, int& a)
{
    j = jour;
    m = mois;
    a = an;
}

void Date::afficher () const
{
   cout << jour << "/" << mois << "/" << an;
}
```


1er Exemple : classe Date (3/3)

- Création de 2 instances :
  - 1 statique : hier
  - 1 dynamique : ptjour
- Programme principal :

```
int main ()
{
   Date hier ;
   Date * ptjour = new Date;

   if (hier.initialiser (4, 11, 1992))
      hier.afficher();
   if (ptjour ->initialiser (2,10,2000))
      ptjour->afficher();
}
```

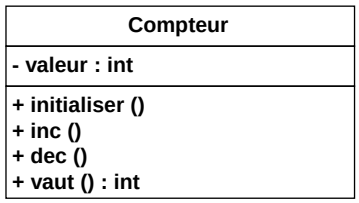


Affichage:

```
4/12/1992
2/10/2000
```

2ème Exemple : Compteurs (1/3)

- Création d'une classe Compteur :
  - 1 attribut entier : valeur
  - 4 méthodes :
    - ➔ initialiser le compteur à zéro
    - ➔ incrémenter
    - ➔ décrementer
    - ➔ lire la valeur courante

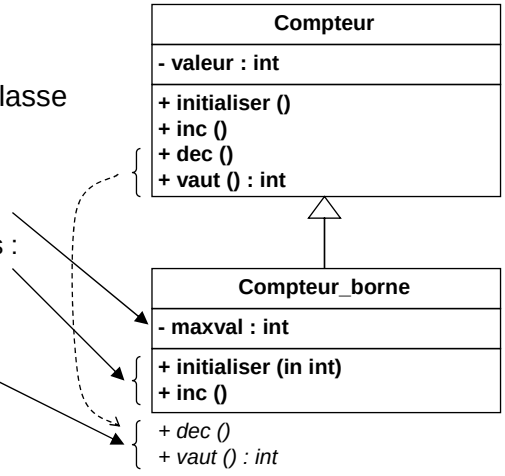



```
class Compteur
{
  private:
    int valeur;
  public:
    void initialiser ();
    void inc ();
    void dec ();
    int vaut ();
};
```

```
void Compteur::initialiser ()
{   valeur = 0;   }

void Compteur::inc ()
{   ++valeur;   }

void Compteur::dec ()
{   if (valeur >0)  --valeur;   }

int Compteur::vaut ()
{   return valeur;   }
```


2ème Exemple : Compteurs (2/3)

- Création d'une nouvelle classe Compteur_borne dérivée de Compteur :
  - 1 attribut ajouté : maxval
  - 2 méthodes surchargées :
    - ➔ initialiser le compteur
    - ➔ incrémenter
  - 2 méthodes héritées :
    - ➔ décrémenter
    - ➔ lire la valeur courante



NB: l'attribut valeur est également hérité mais non accessible

2ème Exemple : Compteurs (3/3)

```
class Compteur_borne : public Compteur
{
  private:
    int maxval;
  public:
    void initialiser (int);
    void inc ();
};

void Compteur_borne::initialiser (int max)
{
    Compteur::initialiser ();
    maxval = max;
}

void Compteur_borne::inc ()
{
    if (vaut ()<maxval)
        Compteur::inc ();
}
```

On ne peut pas écrire :
valeur = 0
car valeur est inaccessible !!!

On ne peut pas écrire :
if (valeur<maxval)
++valeur;
car valeur est inaccessible !!!

Cours Info 2A (1/4) Introduction à la POO et au C++ 29