Module C++

FIP 1A CNAM

Séance 3

Programmer en C++

LES 4 PILIERS DE LA POO

LES 4 PILIERS DE LA POO

HÉRITAGE

L’Héritage:

Définition

- C’est une sorte de relation entre deux classes où une classe implémente toutes les caractéristiques membres et comportements de l’autre (à l’exception des membres privés).
- L’héritage représente un concept de généralisation/spécialisation. Une classe mère est la généralisation de ses classes enfants, et inversement, les classes enfants sont des spécialisations de la classe mère.

L’Héritage:
Utilisation

- On utilise l’héritage principalement pour partager du code entre différents types d’objets.
- Plus particulièrement pour les cas suivants:

- On souhaite factoriser, c’est-à-dire regrouper des attributs et/ou comportements entre plusieurs types d’objets sans avoir à les réécrire.

- On veut étendre une classe en lui ajoutant des caractéristiques comme des données ou des comportements.

- On veut pouvoir manipuler des objets de classes différentes de la même manière. On parle ici « d’abstraction ».

LES 4 PILIERS DE LA POO
L'ABSTRACTION

L’abstraction

- C’est une notion à la fois philosophique et mathématique.
- Elle consiste à expliquer que pour décrire des termes, on peut utiliser des concepts plus génériques ou plus spécifiques. Cela revient à donner plus ou moins de détails.
- En POO, c’est représenté par les niveaux d’héritage.
- Cela revient à masquer les détails dont on n’a pas besoin dans un certain contexte.

LES 4 PILIERS DE LA POO

L'ENCAPSULATION

L’encapsulation : définition

- En programmation, l’encapsulation désigne le principe de regrouper des données brutes avec un ensemble de routines permettant de les lire ou de les manipuler. Ce principe est souvent accompagné du masquage de ces données brutes afin de s’assurer que l’utilisateur ne contourne pas l’interface qui lui est destinée. L’ensemble se considère alors comme une boîte noire ayant un comportement et des propriétés spécifiés.
- L’encapsulation est un pilier de la programmation orientée objet, où chaque classe définit des méthodes ou des propriétés pour interagir avec les données membres[…]

Source: Wikipedia

L’encapsulation: rappels

1. public: Les membres sont accessibles depuis tout le programme.
2. protected: Les membres sont accessibles aux classes dérivées.
3. private: Les membres sont uniquement accessibles à l’intérieur de la classe.

/!\ On ordonne généralement les membres d’une classe dans cet ordre /!\

Les classes « amies » (friend)

- En théorie, les membres privés et protected sont uniquement accessibles au sein de la classe ou également à ses classes enfant.
- Une exception existe. Une classe peut déclarer une classe comme son « amie ».
- Ainsi, elle lui donne accès à tous ses membres privés.

L’encapsulation : utilité

- Elle permet d’exposer une interface sans partager les mécanismes utilisés pour réaliser le traitement de l’information.

=> Exemple de la machine à café:

L’utilisateur veut recevoir un café contre une pièce. En tant que fournisseur, vous fournissez une interface à l’utilisateur, mais il vous appartient de maîtriser les mécanismes internes.

LES 4 PILIERS DE LA POO

LE POLYMORPHISME

Le polymorphisme statique (ad hoc)

- Aussi appelé « surcharge » (overloading) c'est un mécanisme qui permet de définir plusieurs versions d'une même fonction et de choisir la version à appeler. Il/elle peut être :

  - statique: le choix est effectué à la compilation, en fonction du type statique et du nombre d'arguments précisé.

  - dynamique: le choix est fait à l'appel en fonction du type dynamique des objets à l'exécution.

Le polymorphisme paramétrique

- C’est la partie de C++ liée aux templates.
- Aussi appelé « programmation générique » (en C# notamment)
- Consiste à définir des algorithmes identiques qui peuvent effectuer un traitement sur des types de données différents.
- La fonction de tri de la STL est un bon exemple. Avec un moyen d’itérer sur la collection et un moyen de comparaison entre les objets, l’algorithme est capable de trier n’importe quelle collection d’objets.

Le polymorphisme par sous-typage

- C’est l’implémentation du fait que des comportements implémentés dans une classe mère peuvent être ré-implémentés dans les classes enfants.
- Par exemple, des objets représentant des formes peuvent calculer leur aire, mais ne le font pas de la même manière. Cf. TP2.

Programmer en C++

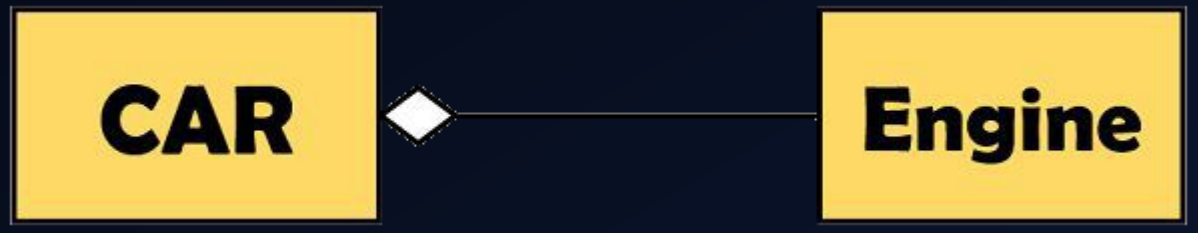
QUELQUES NOTIONS LIÉES À LA POO

QUELQUES NOTIONS LIÉES À LA POO: AGRÉGATION ET COMPOSITION

L’agrégation

- C’est une sorte de relation entre deux classes où une classe implémente des instances de l’autre.
- Une classe est en relation d’agrégation avec une autre lorsqu’elle a un attribut qui en est une instance ou une collection d’instances.
- Dans une relation d’agrégation, l’enfant peut vivre sans le parent.

Ex: Une classe « PromotionCnam » possède des instances de la classe « Étudiant », mais à la destruction de la classe « PromotionCnam », les instances de la classe « Étudiant » continuent d’exister.



La composition

- C’est une sorte de relation entre deux classes où une classe implémente des instances de l’autre, comme pour l’agrégation.
- Toutefois, c’est une relation plus forte que l’agrégation, car les éléments reliés entre eux n’existent pas l’un sans l’autre.

Ex: Une classe « Bâtiment » possède des instance de la classe « Pièce », et les pièces ne peuvent exister en dehors du bâtiment.

Agrégation vs. Composition

- Pour l’agrégation, l’enfant peut exister indépendamment du parent, contrairement à la composition.
- Le type de relation de l’agrégation est « possède… », le type de la composition est « fait partie de… »
- Dans le cas d’une agrégation, l’enfant peut avoir plusieurs parents, pas dans le cas d’une composition.
- L’agrégation est une relation faible, alors que la composition est une relation forte car elle a plus de contraintes.

QUELQUES NOTIONS LIÉES À LA POO:

LE COUPLAGE

Le couplage

- Le couplage entre deux composants est un indicateur du volume d’informations qu’ils communiquent entre eux.
- On parle de couplage « fort » ou « serré » s’ils échangent beaucoup de données entre eux.
- On parle de couplage faible si les composants sont indépendants ou s’ils échangent un minimum de données entre eux.

Les niveaux de couplage

Selon ~~Wikipedia~~ Pressman[1], il existe sept niveaux de couplage, du plus faible au plus fort :

1. **Sans couplage** : pas d’échange d'information.
2. **Par données** : échanges de l'information par des méthodes utilisant des arguments (paramètres) de type simple (nombre, chaîne de caractères, tableau).
3. **Par paquet** : les composants échangent de l'information par des méthodes utilisant des arguments de type composé (structure, classe).

Les niveaux de couplage

4. **Par contrôle** : les composants se passent ou modifient leur contrôle par changement d'un drapeau (verrou).
5. **Externe** : les composants échangent de l'information par un moyen de communication externe (fichier, pipeline, lien de communication).
6. **Commun (global)** : les composants échangent de l'information via un ensemble de données (variables) commun.
7. **Par contenu (interne)** : les composants échangent de l'information en lisant et écrivant directement dans leurs espaces de données (variables) respectifs.

Problèmes d’un couplage fort

- Plus le couplage entre deux objets est fort, plus la modification de l’un entraîne la modification de l’autre, et plus les objets sont difficilement testables.
- Il est difficile d’estimer l’étendue des modifications à réaliser pour modifier un composant.
- La structure du programme est très rigide, l’évolution du programme est difficile et coûteuse.

La solution au couplage fort

- Pour éviter un couplage fort, on préconise un couplage faible, ce qui se traduit par l'utilisation de mise en place de standards de communication entre les composants.
- Les échanges doivent imposer le moins de contraintes possibles aux composants impliqués.

QUELQUES NOTIONS LIÉES À LA POO: QUELQUES MOTS CLÉS

Virtual

- Déclarer une fonction « virtual » permet de ne faire le lien entre la fonction et son implémentation qu'à l'exécution.
- Avec ce mot-clé, la version de la fonction à appeler n'est plus choisie en fonction du type de pointeur à la compilation (« early binding »), mais en fonction du type réel de l'objet à l'exécution (« late binding »).

Les variables « static »

- static: Les membres d’une même classe partagent tous cette instance du membre.
- Le membre n’est donc pas stocké dans l’instance de l’objet, mais dans le tableau des symboles.
- Selon le standard et le compilateur, il peut être obligatoire de l’assigner en dehors de la définition de la classe.

Programmer en C++

MÉTHODES NÉCESSAIRES AU BON FONCTIONNEMENT D'UNE CLASSE

LES FONCTIONS AUTOMATIQUEMENT CRÉES EN C++

Les fonctions automatiquement crées en C++

```
class Empty{};
```

Revient à écrire :

```
class Empty {
    public:
        Empty() { ... } // constructeur par défaut
        Empty(const Empty& rhs) { ... } // constructeur par copie
        ~Empty() { ... } // destructeur
        Empty& operator=(const Empty& rhs) { ... } // operateur d'assignation par copie
};
```

Les fonctions automatiquement crées en C++

- Les fonctions précédemment mentionnées ne sont générées que si elles sont appelées, comme dans l'exemple ci-dessous.

```
{
    Empty e1;    // constructeur par défaut
}                // destructeur
Empty e2(e1);    // constructeur par copie
e2 = e1;         // operateur d'assignation par copie
```

- Il en faut donc peu pour qu'elles soient instanciées.

Les opérateurs d’assignation

- Le constructeur par copie, souvent appelé **X(X&)** (« X of X ref »), est essentiel pour pouvoir passer par valeur un type défini par l’utilisateur.
- L’opérateur « = » (égal) ou constructeur par copie est également essentiel au fonctionnement de tout programme.
- Si on ne le définit pas nous-même, le compilateur en définira un par défaut.

Les fonctions automatiquement crées en C++: Que font-elles?

- Les constructeurs par défaut appellent les constructeurs par défaut de tous les membres non statiques.
- L'opérateur d'assignation par défaut instanciera une copie de l'objet passé en paramètre à l'endroit occupé par l'objet sur lequel on appelle l'opération.
- Le destructeur par défaut libérera l'espace mémoire occupé par l'objet (NB: mais pas les espaces situés aux adresses des pointeurs membres !!!)

Les fonctions automatiquement crées en C++

Les fonctions que nous venons de voir sont les seules qui ne sont pas héritées par les classes enfant lors d’un héritage.

Programmer en C++

LES CLASSES ABSTRAITES

Les classes abstraites en C++

- Certaines classes mères existent uniquement pour factoriser des caractéristiques entre d'autres classes filles. Elles ne peuvent pas être instanciées.
- On nomme ces classes des classes abstraites.
- Une classe est dite abstraite si elle ne propose pas d'implémentation pour au moins une de ses méthodes. Une telle méthode est dite « virtuelle pure »
- Les classes dérivées, qui sont concrètes, elles, implémentent ces méthodes qui deviennent des méthodes concrètes.

Les classes abstraites en C++

- Exemple de classe abstraite:

```
class Animal {
    public:
        Animal();
        Animal(int _age) { age = _age; }
        virtual void Manger() = 0;
        virtual void Deplacer(Direction d);
    private:
        int age;
}
```

- Essayer de l'instancier provoquera une erreur de compilation, car la méthode « Manger() » a été déclarée comme « virtuelle pure ».

Les classes abstraites

- Exemple de classe abstraite:

```
class Animal {
    public:
        Animal();
        Animal(int _age) { age = _age; }
        virtual void Manger() = 0;
        virtual void Deplacer(Direction d) = 0;
    private:
        int age;
}
```

- Toutes les méthodes de cette classe sont « virtuelles pures », la classe est donc elle aussi « virtuelle pure ». C’est ce qui se rapproche le plus d’une interface (du C# ou du Java) en C++.

Programmer en C++

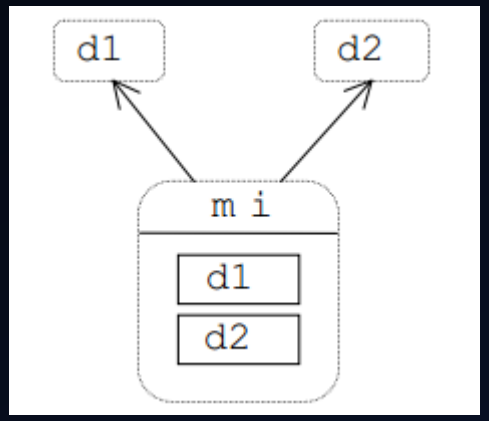
L'HÉRITAGE MULTIPLE

L’héritage multiple

- L’héritage multiple est dans le C++ dès sa première version. La rétrocompatibilité entre les versions étant non négociable, c’est un mécanisme qui est toujours présent.
- Le recours à ce principe est toujours sujet à débat.

L’héritage multiple

Ici, la classe mi hérite des classes d1 et d2, l’ensemble des données que ces classes comporte est donc copié dans la classe mi.



Maintenant, imaginez que les deux classes d1 et d2 héritent de la même classe de base.

L’héritage multiple

Les objets de la classe mi comportent donc deux fois les données de la classe base.

Le problème principal est que ça introduit une complexité liée à l’ambigüité entre les deux versions.

Ce problème est appelé le problème de l’héritage « en diamant ».