Notes de Cours INF 1010

Olivier Sirois

2018-01-01

Contents

Plan de cours

Chapter 1

Séances de Cours 1

1.1 Concepts de base orienté-objet et constructeur/destructeur

dogme Tout est en objet, le development des modeles sont fait à partir ' d'objet '. ceux-ci sont représenter comme étant une abstraction d'une partie du problème. On essaie de conceptualiser nos objets de la même manière que notre problème est représenter afin que ces objets puisse interagir entre eux de la même manière que le vraie problème.

Nos objets contiennent des **attributs** et des **méthodes**. Des attributs sont des paramêtre représentant certaines caractéristique de l'objet en question. Comme par exemple le poids et l'age d'une personne, ou la couleur d'une voiture.. Tandis qu'une méthodes est un action que cet objet en question peut faire. comme par exemple afficher tous ses attributs, faire déplacer une personne dans une simulation ou bien mettre la voiture en marche. Une méthode peut être définis comme étant des fonctions propres à chaque objet (classe)

Nos objets sont calqués à partir d'une **classe**. Cette classe est comme un 'blueprint' d'un objet. Cette classe montre explicitement tous les attributs et méthodes d'une classe. Nos objets ont des relations avec leur classe comme par exemple deux marques de voitures ferait parti de la classe 'voiture', la différence ferait parti des différents attributs.

Nos classes donne aussi de l'info sur la **visibilité** des attributs et méthodes. La visibilité est un manière de décrire d'ou on peut accéder à l'information. Comme par exemple, il est intéressant d'avoir des attributs qui ne peuvent être modifié pour la classe elle-même, cela fera en sorte qu'on a beaucoup moins de risque d'accidentellement modifié son contenu par accident.

- public : tout le monde peut accéder

- privé : seulement les fonctions dans la classe elle-même peut accéder aux éléments privé

Certaines méthodes d'un classe sont publique (pour qu'on puisse les appeller de notre programme). Ces méthodes sont appelés **interface** de notre programme. Une capacité intéressante est de pouvoir définir des fonctions qui sont privés, des fonctions internes que les méthode publique peuvent appeller pour accomplire leur tâches.

la relation entre une classe et son objet est la même que qu'une variable et son type.

INSERT SLIDE 8

Pour initialiser nos objets, on utilise le **constructeur**. C'est une méthode spécial qui sert à faire la création de notre objet en mémoire. On s'en sert généralement pour donner nos paramêtre à nos attributs dans notre objet lors de l'initialisation de notre objet. Le constructeur par défaut prend aucun paramêtre mais n'initialise aucun valeur dans nos attributs. On peut redéfinir un constructeur en utilisant une fonction ayant le même nom que la classe. Notre redéfinition peut avoir des valeurs comme arguments que nous pouvons ou non placer dans nos attributs.Normalement, notre constructeur doit être publique.

INSERT SLIDE 9 et 15

Par convention, on utilise un underscore (_) pour représenter les attributs privés de notre classe. Sa fait en sorte que nos attributs sont beaucoup plus visible lors d'inspection de notre programme.

En générale, les méthodes sont implémentés séparément de la définition de classe.c-a-d, que la définition de nos fonction ne se font pas dans les brackets. On doit par contre quand même faire la déclaration de nos fonctions dans la classe, mais la définition peut se faire à l'extérieur des brackets. On utilise le scope operator (::) pour dire que notre définition appartient à une classe spécifique

INSERT SLIDE 17

Normalement, on doit être capable de faire certaine manipulation sur des objets:

- On peut obtenir la valeur des attributs
- On peut chagner les valeurs
- On peut demander à un objet d'en créer un autre
- On peut demander à un objet d'en manipuler un autre

On utilise le dot operator (.) pour utiliser des méthodes d'un objet

INSERT SLIDE 19

Tout méthode à un paramêtre implicite. qui correspond à l'objet sur lequel elle est appliqué. les autre paramêtre qui aparaissent sont considérer explicites. (Pour le compilateur)

INSERT SLIDE 21

l'interface contient:

- Constructeur
- fonction de modification (mutateurs)
- fonction d'accès
- autres fonctions utiles

encapsulation On n'a pas toujours accès directement aux attributs d'un objet. On modifie toujours la valeur des attributs à l'aide d'une méthode, jamais directement. En résumé, on ne peut manipuler l'état d'un objet que par le biais des méthodes qui sont définies par sa classe (interface)

constructeur En générale, nos constructeur ne recoit aucun paramêtre et par lui-même donne des valeurs aux attributs On peut aussi avoir des constructeur qui prennent certains argument et qui les passe ensuite aux différent attributs.

Un constructeur par défaut est appelé lorsqu'on utilise pas de parenthèse à la déclaration de notre objet, il appel le constructeur par défaut à ce moment. Il le faut aussi lorsqu'on déclare un array d'objet.

destructeur Une des grandes différence entre le c++ et le java. Le destructeur est une méthode qui est appelé lorsqu'un objet est détruit. Dans le cas de Java, la JVM à un 'garbage collector' qui se charge automatiquement de la gestion de mémoire, c-à-d, de libérer les données qui ne seront plus utilisé. en C++, C'est notre responsabilités de faire la gestion mémoire. Normalement, on fait sa avec le destructeur, c'est la dedans qu'on fait du ménage dans notre mémoire. le destructeur à le même nom que le constructeur, sauf qu'il est précédé par un **tilde**. dans certain cas, on a rien besoin de faire.

Si l'objet est une variable locale, il sera détruit à la sortie de cette fonction. Par contre, s'il est défini dans main, il sera détruit à la fin du programme. Si l'objet est dynamique, le destructeur est appelé lorsqu'on exécute delete sur cet objet.

(plus tard)

1.2 Allocation de mémoire dynamique

En c++ on a le choix de comment on veut gérer la mémoire. On peut le faire automatiquement, c-à-d, laisser le compilateur gérer la destruction des données. Mais on peut aussi le faire nous-même. Ces deux méthodes ont quand même des grosse différences dans leur fonctionnement.

Automatique Dans le cas de la gestion automatique, tous nos variables créer sont storer dans le **stack**. C'est beaucoup plus rapide, mais petit. De plus, on ne peut pas choisir quand on veut détruire nos données, alors pour une programme ou on a beaucoup de données à sauver, on peut rapidement dépasser la grosseur du stack si on créer toujours des objets mais en ne pouvant pas les détruires.

Dynamique Dans le cas de la gestion dynamique, tous nos variables créer sont storer dans le **heap**. le heap est beaucoup plus gros que le stack (des Gb vs quelques MB), mais la gestion de la heap est beaucoup plus strict et certains overhead peut être associer à la création des données. Par contre, cest essentiel si notre programme créer beaucoup d'objet et prend beaucoup de place en mémoire.

Quand on ne sait pas combien de mémoire on a besoin, il est préférable de toujours utiliser la gestion de mémoire dynamique, même si elle est moins performante.

Normalement, quand on utilise la gestion dynamique, on se sert toujours de **pointeurs**. Il est important de prendre l'habitude de toujours initialiser nos pointeurs, au nullptr si on n'a pas de valeur d'initiation.

l'utilisation d'un pointeur sur nos objets modifie quand même les opérateurs qu'on a besoin pour accéder à ses champs. Dans le cas des méthodes, on utilise l'opérateur 'fleche', soit (-¿) pour accéder aux méthodes et aux attributs de notre objet.

Dans le cas de la gestion dynamique, on utilise les commandes **new** et **delete** pour faire l'allocation de mémoire et la désallocation de mémoire. Il est très important d'avoir un delete sur chacune des variable qu'on initialise avec new sinon une fuite de mémoire est imminente...

INSERT SLIDE 17

Dans le cas d'un delete sur un array, il faut utiliser delete [] pour appliquer delete sur tous les éléments d'un array et non seulement sur l'élément que le pointeur pointe (le premier).