Rappel des notions d'héritage, classes abstraites et interfaces[1]

[1] Ce chapitre est un bref rappel des notions étudiées dans le cours ift1170. Pour un cours complet, voir:

http://www.iro.umontreal.ca/~lokbani/ift1170

puis cliquez sur le lien "notes de cours"

Héritage

1. notion de relation

Héritage est une relation **EST-UN**

un chat est un animal.
une voiture est un véhicule.

Objet membre est une relation **A-UN**

une voiture a quatre roues.
une voiture a un moteur.

2. intérêt

- vision descendante => la possibilité de reprendre intégralement tout ce qui a déjà été fait et de pouvoir l'enrichir.

- vision ascendante => la possibilité de regrouper en un seul endroit ce qui est commun à plusieurs.

3. utilisation

- vers le haut (en analyse O.O.) => on regroupe dans une classe ce qui est commun à plusieurs classes. Dans la classe Véhicule, on regroupe les caractéristiques communes aux Camions et aux Automobiles.

- vers le bas (lors de la réutilisabilité) => la classe de base étant définie, on peut la reprendre intégralement pour construire la classe dérivée. La classe Véhicule étant définie, on peut la reprendre intégralement, pour construire la classe Bicyclette.

4. hiérarchisation

- la classe dont on dérive est dite CLASSE DE BASE.

- les classes obtenues par dérivation sont dites CLASSES DÉRIVÉES.

5. classe dérivée

n'est qu'un modèle particulier de la classe de base.

- contient les données membres de la classe de base,

- peut en posséder de nouvelles,

- possède (à priori) les méthodes de sa classe de base,

- peut redéfinir (masquer) certaines méthodes,

- peut posséder de nouvelles méthodes.

6. syntaxe

protection class classe_dérivée **extends** classe_de_base {/* etc. */}

protection: droits d'accès attribués au niveau de la classe. **public** ou bien **néant** (i.e. rien).

7. constructeur

- la classe dérivée doit prendre en charge la construction de la classe de base.

- Cette procédure doit prendre en compte de la présence du constructeur dans la classe de base et/ou dans la classe dérivée.

8. droits d'accès

Les droits d'accès protègent les données et les méthodes, et réalisent aussi l'encapsulation.

Les droits d'accès sont accordés aux fonctions membres, ou aux fonctions globales.

L'unité de protection est la classe: tous les objets de la classe bénéficient de la même protection.

- un membre **public** est accessible à toutes les classes,
- un membre **«rien»** est accessible à toutes les classes du même paquetage,
- un membre **private** n'est accessible qu'aux fonctions membre de la classe.

Si les membres de la classe de base sont :

- public ou « rien » : les membres de la classe dérivée auront accès à ces membres (champs et méthodes),

- private : les membres de la classe dérivée n'auront pas accès aux membres privés de la classe de base.

En plus des 3 niveaux de protection (public, « rien » et private), il existe un 4^e niveau de protection : **protected**. Un membre de la classe de base déclaré protected est accessible à ses classes dérivées ainsi qu'aux classes du même paquetage.

9. Phases d'initialisation d'un objet de la classe de base et de la classe dérivée

Le tableau suivant représente la création d'un objet à partir des classes de base (A) et dérivée (B).

un objet de la classe de base A	Un objet b de la classe dérivée B
allocation mémoire pour un objet du type A	allocation mémoire pour un objet du type B (donc A+B)
initialisation par défaut des champs	initialisation par défaut des champs (A+B)
initialisation explicite des champs	initialisation explicite des champs hérités de A
exécution des instructions du constructeur de A	exécution des instructions du constructeur de A
	initialisation explicite des champs hérités de B
	exécution des instructions du constructeur de B

10. Redéfinition et surdéfinition des fonctions membres

Surdéfinition	Redéfinition
même nom, mais signature différente. Peu importe le type de retour.	même nom, même signature, même type de retour.
cumuler plusieurs méthodes ayant le même nom.	substitution d'une méthode par une autre.

11. Typage statique vs. typage dynamique

- type statique d'une variable: type à la compilation, type déclaré.

- type dynamique d'une variable: type à l'exécution, type en mémoire.

12. Compatibilité entre objets d'une classe de base et objets d'une classe dérivée

Un objet d'une classe dérivée peut toujours être utilisé au lieu d'un objet de sa classe de base.

Polymorphisme

1. Faculté qu'on des objets différents de réagir différemment en réponse au même message.

- Marcher sur la queue d'un chat => il miaule,

- Marcher sur la queue d'un chien => il aboie,

2. Même nom de fonction, plusieurs implantations:

Fonction: opération addition (+)

addition des nombres entiers,

$$1 + 2 = 3$$

addition des nombres complexes,

$$1+2i + 3+4i = 4+6i$$

3. Polymorphisme permet de manipuler des objets sans connaître (tout a fait) le type.

4. Se traduit par:

- la compatibilité par affectation entre un type classe et un type ascendant,

- la ligature dynamique des méthodes.

5. Polymorphisme redéfinition et surdéfinition.

6. Conversion explicite de référence.

Classes et méthodes finales

1. méthode finale ne peut être redéfinie dans une classe dérivée.

2. classe finale ne peut plus être dérivée.

Classes abstraites

1. Intérêt

- Placer dans une classe abstraite toutes les fonctionnalités que nous souhaitons disposer lors de la création des descendants.

- sert comme classe de base pour une dérivation.

2. C'est quoi au juste?

- ne permet pas l'instanciation des objets.

- peut contenir des méthodes et des champs (qui peuvent être hérités) et une ou plusieurs méthodes abstraites.

- une méthode abstraite est une méthode dont la signature et le type de la valeur de retour sont fournis dans la classe et rien d'autre (pas le corps de la méthode i.e. définition).

3. Syntaxe

```
abstract class A { // etc.}
```

4. Particularités

- une classe ayant une méthode abstraite est par défaut abstraite. Donc pas besoin de "abstract" à ce stade.

- doivent être déclarées "public" sinon pas d'héritage!

- nom d'argument muet doit figurer même s'il est sans intérêt!

- classe dérivée d'une classe abstraite n'est pas obligée de redéfinir toutes les méthodes abstraites. Elle peut même n'en définir aucune. Si c'est le cas, elle reste abstraite.

- classe dérivée d'une classe non abstraite peut être déclarée abstraite.

Interfaces

1. C'est quoi au juste?

C'est une classe abstraite particulière:

- pas de méthodes définies, donc toutes les méthodes sont abstraites.

- aucun champ uniquement des constantes.

2. Intérêt

- la possibilité de représenter, sans les implémenter, un ensemble de comportements.

- d'utiliser en quelque sorte l'héritage multiple, même si cette notion est inexistante en java: Une classe dérivée ne peut hériter que d'une seule classe de base (peut être abstraite), mais elle peut implémenter plusieurs interfaces.

3. syntaxe

protection **interface** nom_interface { // etc.}

protection peut-être public ou rien (paquetage). Si public, tous les membres de l'interface sont de facto "public".

4. Implémentation

```
public interface I {
        void f(int n);
        void g();
}

class A implements I {

        // A doit redéfinir les méthodes f & g prévues dans l'interface
}
```

- on ne peut pas différer l'implémentation.

- une même classe peut implémenter plusieurs interfaces.

- on pourra utiliser des variables de types interface, mais on ne peut pas instancier ces variables à des interfaces. On peut par contre utiliser le polymorphisme pour les affecter à des objets implémentant l'interface.

- interface et classe dérivée est une notion indépendante de l'héritage. Une classe dérivée peut implémenter une ou plusieurs interfaces.

- une constante déclarée dans une interface est vue comme une variable **static final**. On ne peut pas donc modifier sa valeur. Elle est accessible de partout ou dans le paquetage (si l'interface n'est pas public).

- la dérivation d'une interface: une interface peut dériver d'une autre en utilisant pour extends réservé pour l'héritage. En réalité, l'opération a simplement permis de concaténer les déclarations. Les droits d'accès ne sont pas pris en compte ce qui n'est pas le cas lors des héritages des classes.