Chapitre III
Héritage (début)

Chapitre III: Héritage

- A) Extensions généralités
  - Affectation et transtypage
- B) Méthodes
  - Surcharge et signature
- C) Méthodes (suite)
  - Redéfinition et liaison dynamique
- D) Conséquences
  - Les variables
- E) Divers
  - Super, accès, final
- F) Constructeurs et héritage

A) Extension: généralités

- Principe de la programmation objet:
  - un berger allemand est un chien
    - il a donc toutes les caractéristiques des chiens
    - il peut avoir des propriétés supplémentaires
    - un chien est lui-même un mammifère qui est lui-même un animal: hiérarchie des classes
  - On en déduit:
    - Hiérarchie des classes (Object à la racine)
    - et si B est une extension de A alors un objet de B est un objet de A avec des propriétés supplémentaires

Extension: généralités

Quand B est une extension de la classe A:

- Tout objet de B a toutes les propriétés d'un objet de A (+ d'autres).
- Donc un objet B peut être considéré comme un objet A.
- Donc les variables définies pour un objet de A sont aussi présentes pour un objet de B (+ d'autres). (Mais elles peuvent être *occultées*)
- Idem pour les méthodes : Les méthodes de A sont présentes pour B et un objet B peut définir de nouvelles méthodes.
- Mais B peut *redéfinir* des méthodes de A.

Extension de classe

Si B est une extension de A

- pour les variables:
  - B peut ajouter des variables (et si le nom est identique cela *occultera* la variable de même nom dans A)

    (occulter = continuer à exister mais "caché")
  - *Les variables de A sont toutes présentes pour un objet B*, mais certaines peuvent être cachées
- pour les méthodes
  - B peut ajouter de nouvelles méthodes
  - B peut redéfinir des méthodes (même signature)

Remarques:

- pour les variables
  - c'est le nom de la variable qui est pris en compte (pas le type).
  - dans un contexte donné, à chaque nom de variable ne correspond qu'une seule déclaration.
  - (l'association entre le nom de la variable et sa déclaration est faite à la compilation)
- pour les méthodes
  - c'est la signature (nom + type des paramètres) qui est prise en compte:
    - on peut avoir des méthodes de même nom et de signatures différentes (surcharge)
    - dans un contexte donné, à un nom de méthode et à une signature correspond une seule définition
    - (l'association entre le nom de la méthode et sa déclaration est faite à la compilation, mais l'association entre le nom de la méthode et sa définition sera faite à l'exécution)

Extension (plus précisément)

- Si B est une extension de A (`class B extends A`)
  - Les variables et méthodes de A sont des méthodes de B (mais elles peuvent ne pas être accessibles: `private`)
  - B peut ajouter de nouvelles variables (si le *nom* est identique il y a occultation)
  - B peut ajouter des nouvelles méthodes si la *signature* est différente
  - B redéfinit des méthodes de A si la signature est identique

Remarques:

- Java est un langage typé
  - en particulier chaque variable a un type: celui de sa déclaration
  - à la compilation, la vérification du typage ne peut se faire que d'après les déclarations (implicites ou explicites)
  - le compilateur doit vérifier la légalité des appels des méthodes et des accès aux variables:
    - a.f() est légal si pour le type de la variable a il existe une méthode f() qui peut s'appliquer à un objet de ce type
    - a.m est légal si pour le type de la variable a il existe une variable m qui peut s'appliquer à un objet de ce type

En conséquence:

- Une variable déclarée comme étant de classe A peut référencer un objet de classe B ou plus généralement un objet d'une classe dérivée de A:
  - un tel objet contient tout ce qu'il faut pour être un objet de classe A
- Par contre une variable déclarée de classe B ne peut référencer un objet de classe A: il manque quelque chose!

Affectation downcast/upcast

```
class A{
    public int i;
    //...
}
class B extends A{
    public int j;
    //...
}
public class Affecter{
    static void essai(){
        A a = new A();
        B b = new B();
        //b=a; impossible que signifierait b.j??
        a=b; // a référence un objet B
        // b=a;
        b=(B)a; // comme a est un objet B ok!!
    }
}
```

« Upcasting »

- Si B est une extension de A, alors un objet de B peut être considéré comme un objet de A:
  - A a=new B();
- On pourrait aussi écrire:
  - A a=(A) new B();
- L'upcasting permet de considérer un objet d'une classe dérivée comme un objet d'une classe de base
- Upcasting: de spécifique vers moins spécifique (vers le haut dans la hiérarchie des classes)
- l'upcasting peut être implicite (il est sans risque!)
- attention
  - il ne s'agit pas réellement d'une conversion: l'objet n'est pas modifié

« Downcasting »

- Si B est une extension de A, il est possible qu'un objet de A soit en fait un objet de B. Dans ce cas on peut vouloir le considérer un objet de B
  - A a=new B();
  - B b=(B)a;
- Il faut dans ce cas un cast (transtypage) *explicite* (la "conversion" n'est pas toujours possible –l'objet référencé peut ne pas être d'un type dérivé de B)
- A l'exécution, on vérifiera que le cast est possible et que l'objet considéré est bien d'un type dérivé de B
- downcasting: affirme que l'objet considéré est d'un type plus spécifique que le type correspondant à sa décalration (vers le bas dans la hiérarchie des classes)
- le downcasting ne peut pas être implicite (il n'est pas toujours possible!)
- attention
  - il ne s'agit pas réellement d'une conversion: l'objet n'est pas modifié

Casting

- On peut tester la classe avant de faire du "downcasting":

```
Base sref;
Derive dref;
if(sref instanceof Derive)
    dref=(Derive) sref
```