Le langage Java

Les classes et les objets

I - Principe de modélisation orientée objet

- **Classe**
- **Objet**
- **Principe d'encapsulation**
- **Réutilisation**

Une classe

- Un objet possède des *attributs* (propriétés) et des comportements ou des opérations sur ces attributs, réalisées par des *fonctions membres* (on les appelle aussi *méthodes*)

- On définit une classe pour représenter un ensemble d'objets ayant les mêmes propriétés et des comportements communs.

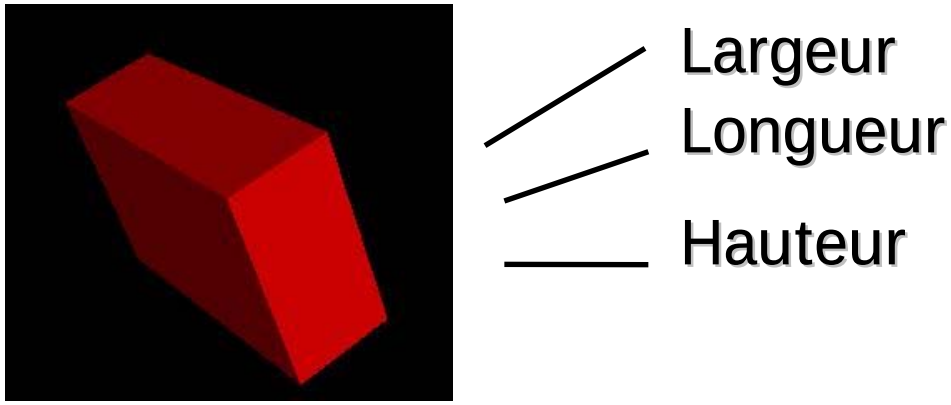
Une classe

- Supposons un logiciel qui doit traiter plusieurs types de cubes, tel que une boîte 10x5x20, de couleur rouge, ou bien un cube 5x5x5 de couleur orange.

- Il faut travailler avec plusieurs types de cubes mais qui possèdent tous les mêmes propriétés et les mêmes opérations (ou comportements) pour modifier ces propriétés.

Une classe (suite)

- Une classe est donc un « super » type que l'on définit, une généralisation d'un objet possédant des propriétés et des opérations (ou comportements).
- Type de données abstrait
- Dans le langage Java, une classe ressemble à une structure que l'on définirait (**`struct type_cube`**) avec des méthodes (ou fonctions membres) pouvant modifier les champs de la structure.



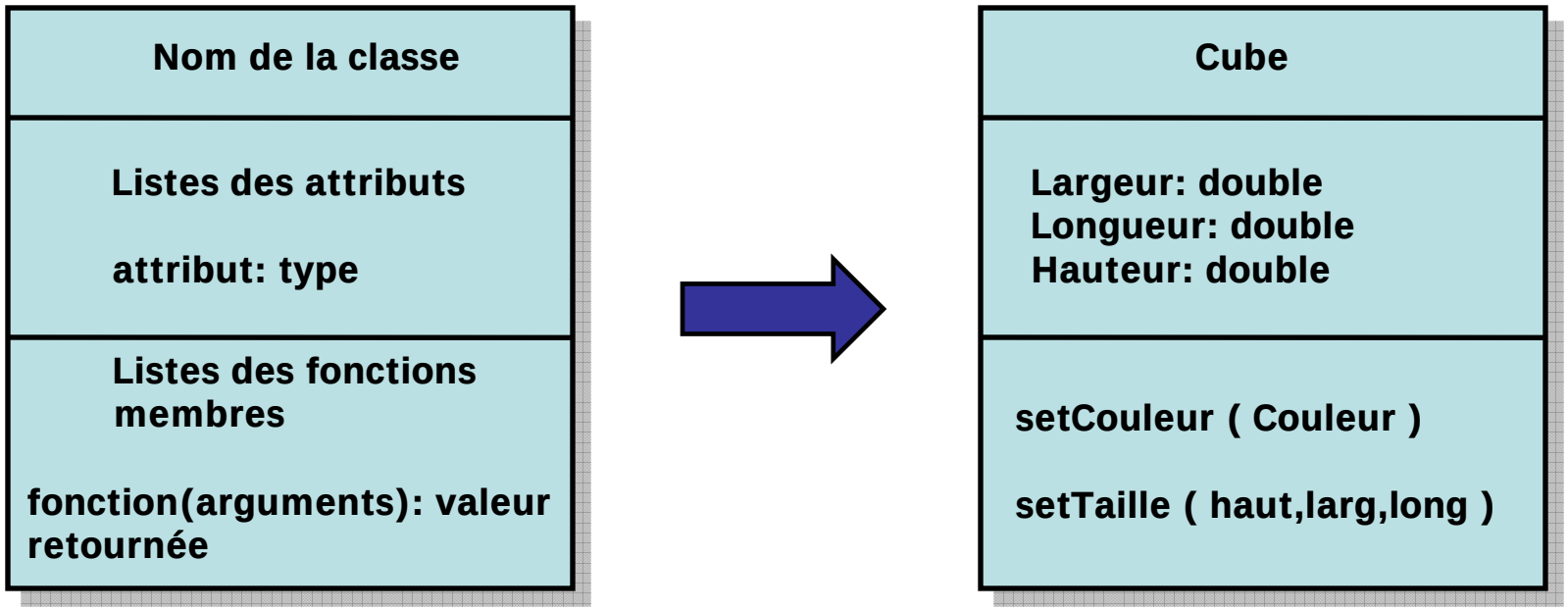
Attributs et opérations d'une classe

- Les propriétés d'un objet sont représentées à l'aide d'attributs.
- Des valeurs sont associées à ces attributs.
  Exemple : Largueur de Cube1 est 10

- Les comportements d'un objet sont représentés à l'aide de fonctions membres (méthodes).

- Une fonction membre est une action ou une transformation qui peut être effectuée sur un objet ou par un objet. Exemple : Pour effectuer un déplacement, il est nécessaire de faire ***translater*** le cube.

Représentation d'une classe



Un objet

Exemple:
Un rectangle 5x10x20 est un objet de classe Cube et un Cube 5x5x5 est un autre objet de la classe Cube.

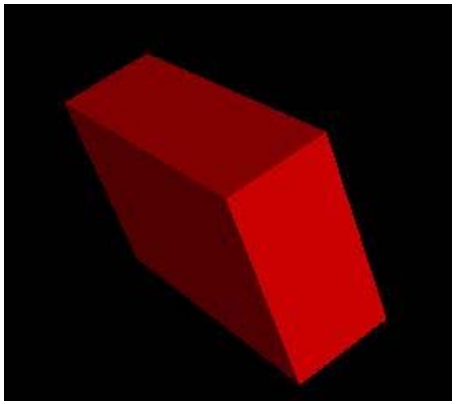
- *Un objet est donc une instanciation d'une classe.*

- *Cet objet possède tous les attributs et toutes les fonctions membres de la classe, mais avec des valeurs d'attributs propres à l'objet.*

Un objet (suite)

Une fois qu'une classe Cube a été définie, il est possible d'instancier des objets de la classe Cube, qui posséderont des attributs associés à des valeurs.

rectangle

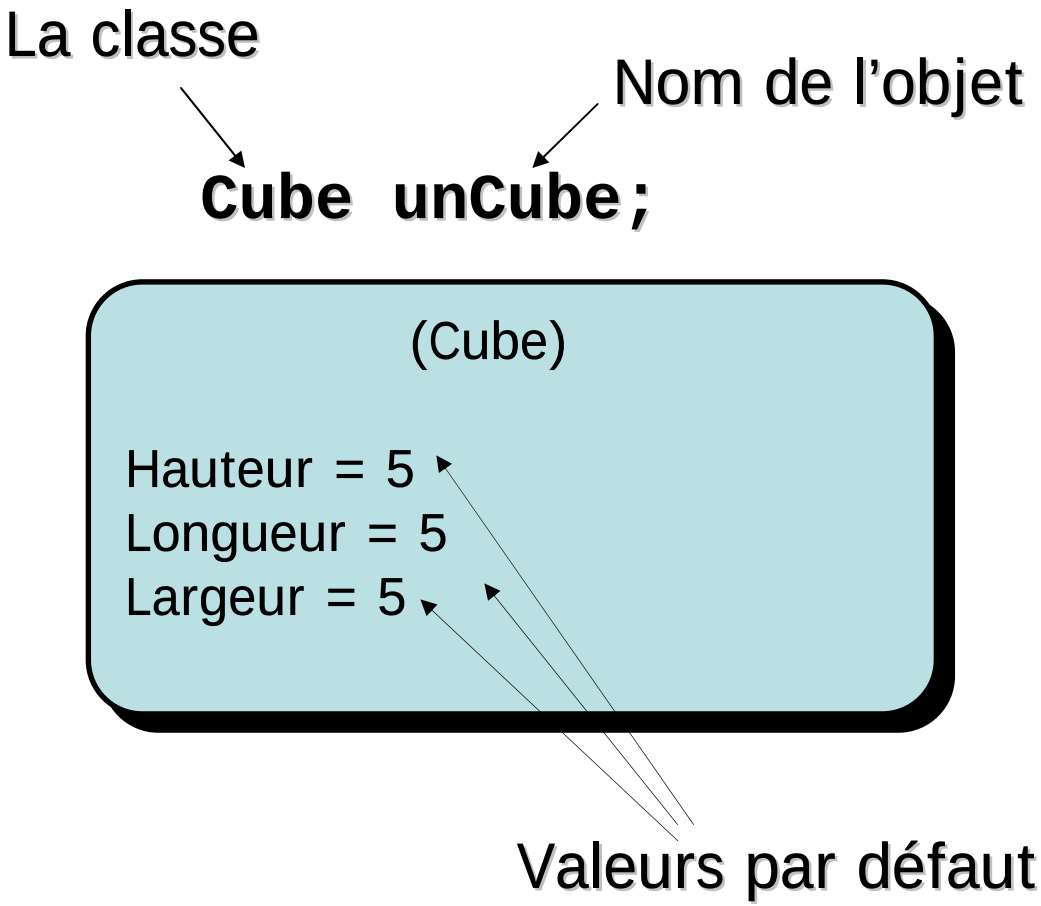


Largeur = 5

Hauteur = 10

ChangerCouleur

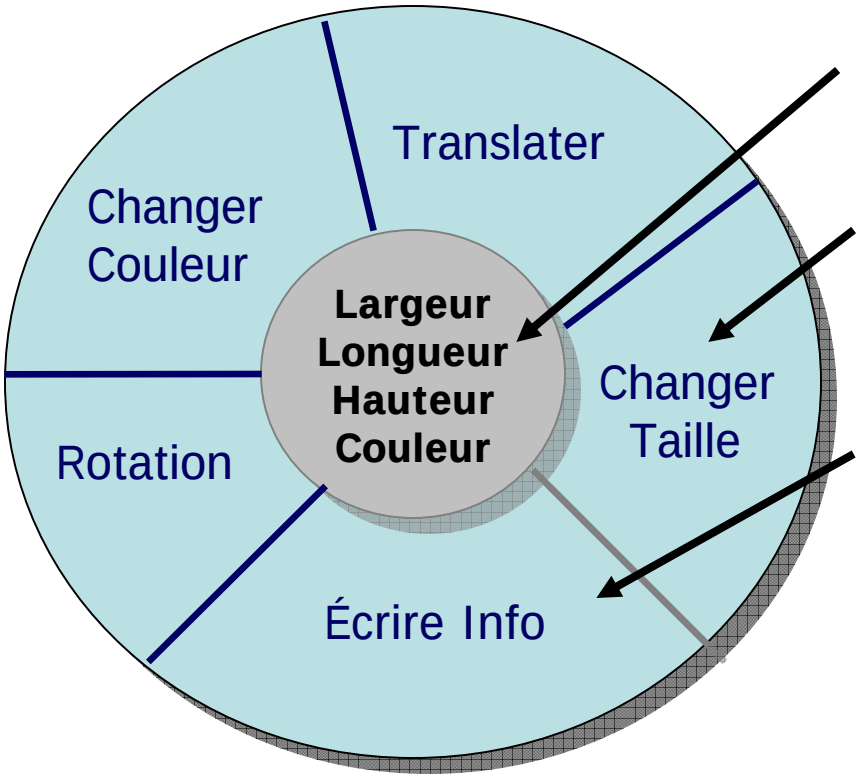
Représentation d'une instance



Programmation par objets

- Principe d'encapsulation (masquage de l'information)
- Types de données abstraits (classe indépendante de son implémentation). L'implémentation de classe peut changer et le client est toujours fonctionnel.
- Réutilisation, réutilisation, réutilisation...
- Réutilisation: agrégation, héritage, généralisation.

Principe d'encapsulation



Les valeurs des attributs peuvent être changées seulement par les méthodes fournies avec l'objet.

L'état de l'objet ne peut être modifié directement par le client.

Les opérations pour afficher les valeurs des attributs pour le client

Définition et implémentation d'une classe

- Classe
- Attributs
- Méthodes
- Principe d'encapsulation et abstraction des données

Classe

- En Java, la définition d'une classe débute toujours en spécifiant ***class,*** suivie du nom de la classe.

  - Toute la classe, c'est-à-dire les attributs et les signatures (entête) de fonctions, est contenue entre des accolades.

Exemple:

```
class Cube
{
   (. . . Définition)
};
```

Attributs

- Pour les protéger, on indique que les attributs seront classifiés *private.*

- Les attributs d'une classe sont des variables ou des références, mais qui ne seront généralement accessibles que par les fonctions membres de la classe. Principe d'encapsulation.

- D'aucune façon les attributs pourront être modifiés ou accédés à l'extérieur de la classe (ex: à partir du main()). Ils seront accessibles à l'aide des fonctions membres.

Méthodes

- Afin de pouvoir manipuler les attributs d'une classe, nous avons recours aux méthodes qui permettent de retourner ou modifier la valeur d'un ou plusieurs attributs.

- Les méthodes qui seront accessibles au client de la classe seront signalées par : « ***`public`*** »

- De ce fait même, les méthodes qui n'ont aucune utilité à l'extérieur de la classe mais qui seront utilisées uniquement par la classe sont « ***`private`*** ».

Méthodes (suite)

Pour la classe Cube, on a :

```
class Cube
{
        private int maLargeur;
        private int maLongeur;
        private int maHauteur;
        private Couleur maCouleur;

        ...
    public void setTaille(float uneLargeur, float uneLongueur,
                          float uneHauteur)
    {
        maLargeur = uneLargeur;
        maLongueur = uneLongueur;
        maHauteur = uneHauteur;
    }
    ...
};
```

Constructeurs

- Constructeurs
- New d'un objet (constructeur)

Le constructeur

- Le constructeur est une méthode publique appelée lors de la création d'un objet (new) qui ne déclare aucun type de retour.
- Le constructeur sert à initialiser un objet lors de sa création.
- Le constructeur par défaut n'a pas de paramètres. S'il n'est pas défini, le constructeur par défaut de la classe de base est utilisé.

Constructeur

```
public Cube(float uneLargeur, float uneLongueur,
            float uneHauteur)
{
     maLargeur = uneLargeur;
     maLongueur = uneLongueur;
     maHauteur = uneHauteur;
}
```

Références

- En Java tous les identificateurs d'objets sont des références (il n'y a pas des pointeurs)
- Il existe des types primitifs (int, float, boolean, etc.) dont les identificateurs ne représentent pas des références
- Il existe des « wrapper » des types primitifs qui sont des objets (Integer, Float, etc.)
- Attention: les chaînes de caractères (String) sont des objets

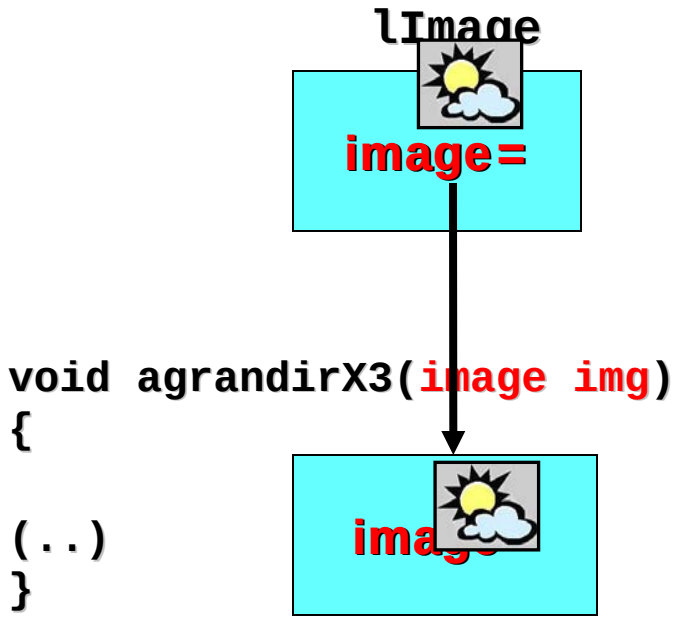
Passage par valeur

- La transmission par valeur transmet la « valeur » du paramètre actuel dans le paramètre formel de la méthode appellée.

```
public void agrandirX3(image img)
{

/*--- modification de l'image transmise a la fonction ----*/
    img.bitmap = imgX2.bitmap;
    img.largeur = imgX2.largeur;
    img.hauteur = imgX2.hauteur; // <- img RECU EN VALEUR,
                                    NE CHANGERA PAS!
}
```

Schématisation du passage par valeur



Le contenu de lImage
est copié dans le paramètre
img

La référence this

- Le mot *this* est un mot réservé pour représenter une référence à l'objet courant.

Surcharges d'opérateurs

- Surcharge des opérateurs binaires, unaires, opérateur assignation

Introduction

La surcharge (ou surdéfinition) d'opérateur est un concept qui permet d'implémenter une fonction ou un opérateur de différentes façons sans en changer la méthode d'appel (signature).

Besoin de la surcharge d'opérateurs

- En Java, il existe des variables de types simples ***int***, ***float***, ***double***, sur lesquelles on peut effectuer des opérations arithmétiques.

- En Java, il n'existe aucun type de base ***String***, d'où le besoin de créer une classe **String** pour traiter les chaînes de caractères.

- Cette classe **String** possède certains opérateurs définis pour un type tel que un entier ou un double (+, =, etc).

Application de la surcharge d'opérateurs

```
String S1 = new("allo ");
String S2 = new("le monde")
String S3 = null;
Char c = '';
S3 = S1 + S2;
S3 = S1.concat("a tous");
c = S2.charAt(1);
s2 = s1
```

Cet exemple est rendu possible grâce à l'utilisation de la surcharge des opérateurs `+` et `=` pour la classe **String**.

Surcharges d'opérateurs

- Il faut bien comprendre le fonctionnement de l'opérateur.
- L'ordre de priorité est conservé.
- Il n'est pas possible de créer de nouveaux opérateurs.
- Il n'est pas possible d'étendre la surcharge

Surcharges de l'opérateur d'assignation (=)

- L'opérateur = demande beaucoup d'attention
  - Égalité de références vs. égalité de contenu

Membres statique

- Un attribut « *static* » n'existe qu'en une seule copie pour tous les objets d'une classe.
- Un attribut « *static* » est un espace commun à tous les objets instanciés.
- L'espace mémoire d'une variable de classe existe même si aucun objet n'a été déclaré.

Méthodes « static »

- Une méthode « *static* » ne peut accéder qu'aux membres « *static* » d'une classe.
- Les règles d'encapsulation demeurent les mêmes.

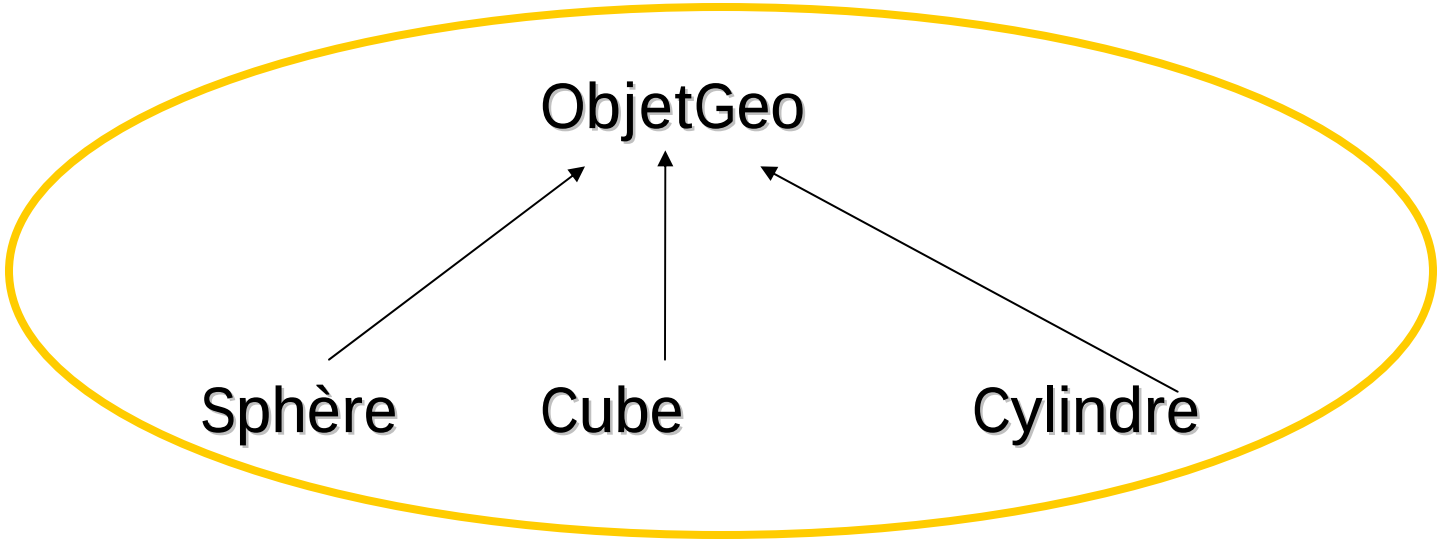
Methode « main »

```
static public void main(String[] args) {
   Class1 c1 = new Class1();
   c1.displayStrings();
}
```

L’héritage

- Concept
- Principe d’héritage
- Private, protected, public
- Surcharge de fonctions
- Constructeurs
- Ordre des appels de constructeurs, destructeurs et agrégat
- Méthodes non héritées
- Dérivations publiques, protégées et privées

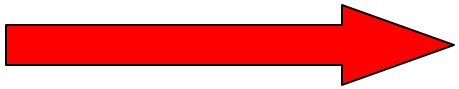
Concept d'héritage



- ObjetGeo: super-classe ou classe de base. Classe très générale.
- Sphère, Cube, Cylindre: sous-classe ou classe dérivée. Classes plus spécifiques.

Principe d'héritage

- Afin de bien représenter une situation, il faut pouvoir montrer les nuances entre des objets, et effectuer certaines généralisations et spécialisations.
- Les classes dérivées sont un mécanisme simple pour définir une nouvelle classe en spécialisant des attributs ou des comportements (méthodes) d'une classe existante.



PARTAGE D'ATTRIBUTS ET DU COMPORTEMENT

Principe d'héritage (suite)

- On peut ainsi utiliser l'héritage pour les besoins de généralisation et de spécialisation
- La classe dérivée hérite des attributs et des fonctions de la classe de base.
- La classe dérivée est plus spécifique que la classe de base en ajoutant ou en re-définissant des attributs et des méthodes.

Principe d'héritage (suite)

- L'héritage est une relation « est un » (IS-A)
  - Un commerce « est un » immeuble
  - Une habitation « est un » immeuble
- Mais il est faux de dire :
  - Un immeuble est un commerce
  - Un immeuble est une habitation

Principe de composition

La composition ou agrégation est une relation « possède un » (HAS-A)

- Un immeuble possède une adresse,
- Un immeuble possède un point3D

Il serait faux de concevoir le logiciel en énoncant que :

- Un immeuble est une adresse (Immeuble est dérivé d'Adresse)

Autres relations

- La relation « utilise un » correspond à l'appel d'une fonction membre d'une classe dont un paramètre est un objet d'une autre classe.

- La relation « connaît un » correspond à une composition par adresse ou une association.

Principe d'héritage (suite)

- En Java, il existe l'héritage simple seulement.

- Dans la définition de la classe dérivée, afin d'utiliser l'héritage, on ajoute le mot clef

  « extends » après le nom de la classe en précisant par la suite quelle est la classe de base.

Ex: **class Sphere extends ObjetGeo**

{…}

Les types d'attributs

Les attributs « ***private*** »

- Lors de l'héritage, les attributs privés de la classe de base restent privés pour les classes dérivées.
- Il faut donc utiliser les ***méthodes publiques*** de la classe de base afin d'accéder à ces attributs.

Les types d'attributs (suite)

Les attributs ou méthodes ***publics***

Lors de l'héritage,

- Les attributs ou méthodes publics de la classe de base seront accessibles par les classes dérivées

- et le resteront pour les clients de la classe dérivée.

Les types d'attributs (suite)

Les attributs ou méthodes « ***protected*** »

Lors de l'héritage:

- Les attributs protégés de la classe de base seront accessibles par les classes dérivées,
- mais ne seront pas accessibles par les clients de la classe dérivée.
- Ce type d'attribut est le plus utilisé lors de l'utilisation de l'héritage.

Redéfinition des fonctions de base

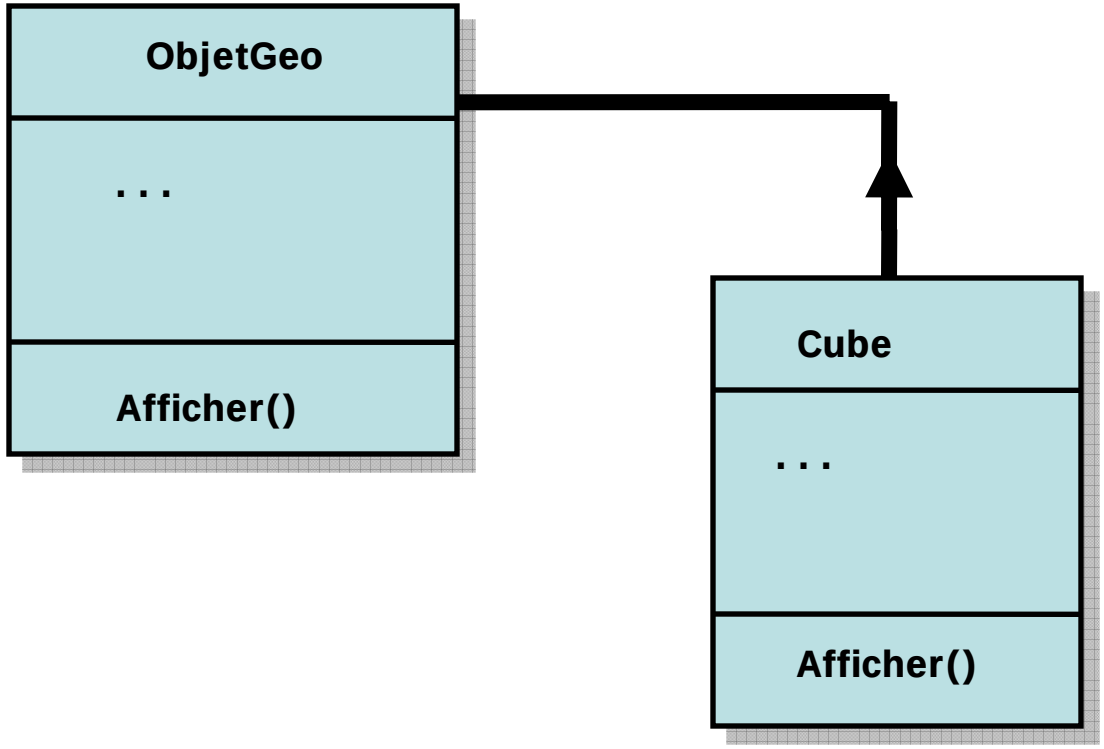
- Les fonctions de la classe de base peuvent être redéfinies dans la classe dérivée.

- Les fonctions redéfinies de la classe de base demeurent accessibles via le mot clef « super »
- Exemple:

```
super.print()
```

Exemple de redéfinition de fonctions de base

Soit la classe ObjetGeo avec sa classe dérivée Cube, toutes les deux possédant une fonction Afficher();



Redéfinition de méthodes

```
… ObjetGeo …
public void Afficher()
{
   System.out.write("Couleur" + maCouleur.ecrire() + "\n");
   System.out.println("Transformation" + maTransformation.ecrire());
}
```

```
… cube …
public void Afficher()
{
    super.Afficher();
    System.out.println(maHauteur + maLongueur + maLargeur);
}
```

Appel de la fonction Afficher() de ObjetGeo

Constructeurs

Lors de la création d'un objet d'une classe dérivée, les constructeurs de la classe de base sont appelés explicitement:

super(..., ..., ...);

Interfaces

- Java permet de déclarer des ensembles de constantes et de signatures de méthodes en utilisant les mots clef «interface » et « implements »

- La syntaxe est:

```
public interface monInterface extends Interface1, Interface2, Interface3 {
    // déclaration de constantes

    // base des logarithmes naturels
    double E = 2.718282;

    // signatures

    public void faitQuelqueChose (int i, double x);
    public int faitAutreChoses(String s);
 }
```

Interfaces (suite)

```
public class maClasse implements monInterface  {

    public void faitQuelqueChose (String s, boolean flag) {
      if (flag)
                System.out.println(s);
    };

    public int faitAutreChoses(int i);
      return(i++);
    };

    public void print() {
      System.out.println("Classe: maClasse");
 }
```