Programmation Orientée Objet : les concepts de base

1. Les classes & les objets

Une classe =

- des « attributs »
- des « méthodes »

} les « membres » de la classe

En Java :

```
public class NomClasse {
    //attributs
    ...
    //constructeurs
    ...
    //autres méthodes
    ...
}
```

Un objet = une instance particulière de la classe

une classe → 1er objet, 2ème objet, 3ème objet, …

- mêmes attributs (valeurs différentes ou non)
- même dictionnaire de méthodes

« lien d'instanciation »

En Java :

```
NomClasse nomObjet;
```

ou

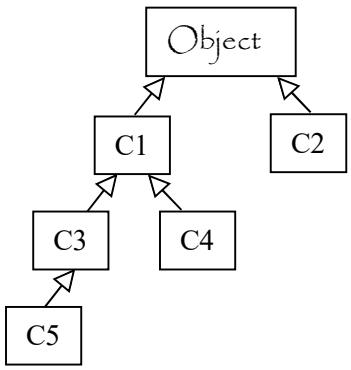
```
NomClasse nomObjet = new NomClasse(...);
```

2. L'héritage

Intérêt :

définir **une classe** (« classe fille » ou « sous-classe ») **à partir d'une autre classe** (« classe mère » ou « super-classe »).

Hiérarchie de classes :



« lien d'héritage »

C3 « hérite » ou « dérive » de C1.

C1, C3 « classes ancêtres » ou « classes ascendantes » de C5.

C3, C4, C5 « classes dérivées » ou « classes héritières » ou « classes descendantes » ou « sous-classes » de C1.

En Java : héritage simple

Propriétés :

- une sous-classe
  - **hérite** des membres de ses ancêtres,
  - peut **ajouter** de nouveaux membres,
  - peut **redéclarer** des méthodes héritées,
- la relation d'héritage est une **relation transitive**,
- toute classe hérite d'une autre classe

  => un **ancêtre commun** implicite = « classe racine » ou « classe de base »

  (en Java : classe Object)

En Java :

```
public class NomSousClasse extends NomSuperClasse {
//nouveaux attributs
        ...
//constructeurs
        ...
//méthodes redéclarées
        ...
//méthodes propres à la classe
        ...
}
```

3. Les niveaux de visibilité

Intérêt :

contrôler l'accès aux membres de la classe ou à une classe (en Java).

Différents niveaux :

- **privé** (private) = accès uniquement dans la classe,
- **protégé** (protected) = accès dans la classe et dans les classes dérivées,
- **public** (public) = accès dans la classe, dans les classes dérivées et dans les autres classes.

4. L'encapsulation

Intérêt :

garantir **l'intégrité des données**.

En pratique :

- les attributs sont déclarés **privés**,
- si nécessaire, des méthodes **publiques ou protégées** pour les manipuler.

5. Les méthodes

Caractéristiques :

- définies pour **tous les objets** de la classe,
- activées **uniquement par les objets** de la classe,
- dans la déclaration de la méthode : l'objet appelant est **implicite.**

Différentes catégories :

- le(s) **constructeur**(s) et le(s) **destructeur**(s),
- les **accesseurs** (« getters ») : accéder à la valeur d'un attribut,
- les **mutateurs** (« setters »): modifier la valeur d'un attribut,
- les autres : faire un traitement.

Les constructeurs & les destructeurs :

- **Rôle :**
  - constructeur = **créer** un objet (allouer un emplacement mémoire) et **initialiser** les attributs,
  - destructeur = **détruire** un objet (libérer l'emplacement mémoire).
- **Caractéristiques :**
  - méthodes en général **publiques,**
  - **au moins un** constructeur et un destructeur par classe (éventuellement hérités).

En Java :

- nom du constructeur = nom de la classe

déclaration :

```
public NomClasse (...) {
    ...
}
```

appel :

```
nomObjet = new NomClasse (...);
```

- si pas de constructeur dans la classe => génération automatique d'un « constructeur par défaut » ou « constructeur sans paramètre »,

- si uniquement un(des) constructeur(s) avec paramètre(s) dans la classe => constructeur obligatoire dans les classes descendantes,
- « ramasse-miettes » (« garbage collector ») : destruction automatique des objets quand ils ne sont plus référencés,
- si traitement particulier à faire avant que l'objet ne soit détruit : redéclarer un finaliseur

```
public void finalize () {
        ...
}
```

au plus un finaliseur par classe

Les autres méthodes :

```
modificateur void nomMethode (...) {
        ...
}
modificateur type_retour nomMethode (...) {
        ...
        return ... ;
}
```

6. L'accès aux membres d'une classe

Règles de validité :

- le membre doit être **visible,**
- le membre doit être **déclaré dans la classe** ou **dans l'un de ses ancêtres**

  => recherche **ascendante** dans la hiérarchie de classes

Objet appelant :

	dans la classe	dans une classe dérivée	ailleurs
accès à un attribut	nomAttribut ou this.nomAttribut	X	X
accès à un constructeur	o dans un autre constructeur : this (...) ; o sinon : X	o dans un constructeur de la sous-classe : super (...) ; o sinon : X	nomObjet = new NomClasse(...) ;
accès à une méthode	nomMethode (...) ; ou this.nomMethode (...) ;	o si la méthode est redéfinie dans la sous-classe : super.nomMethode (...) ; o sinon : nomMethode (...) ; ou this.nomMethode (...) ;	nomObjet.nomMethode (...) ;

Objet autre que l'objet appelant :

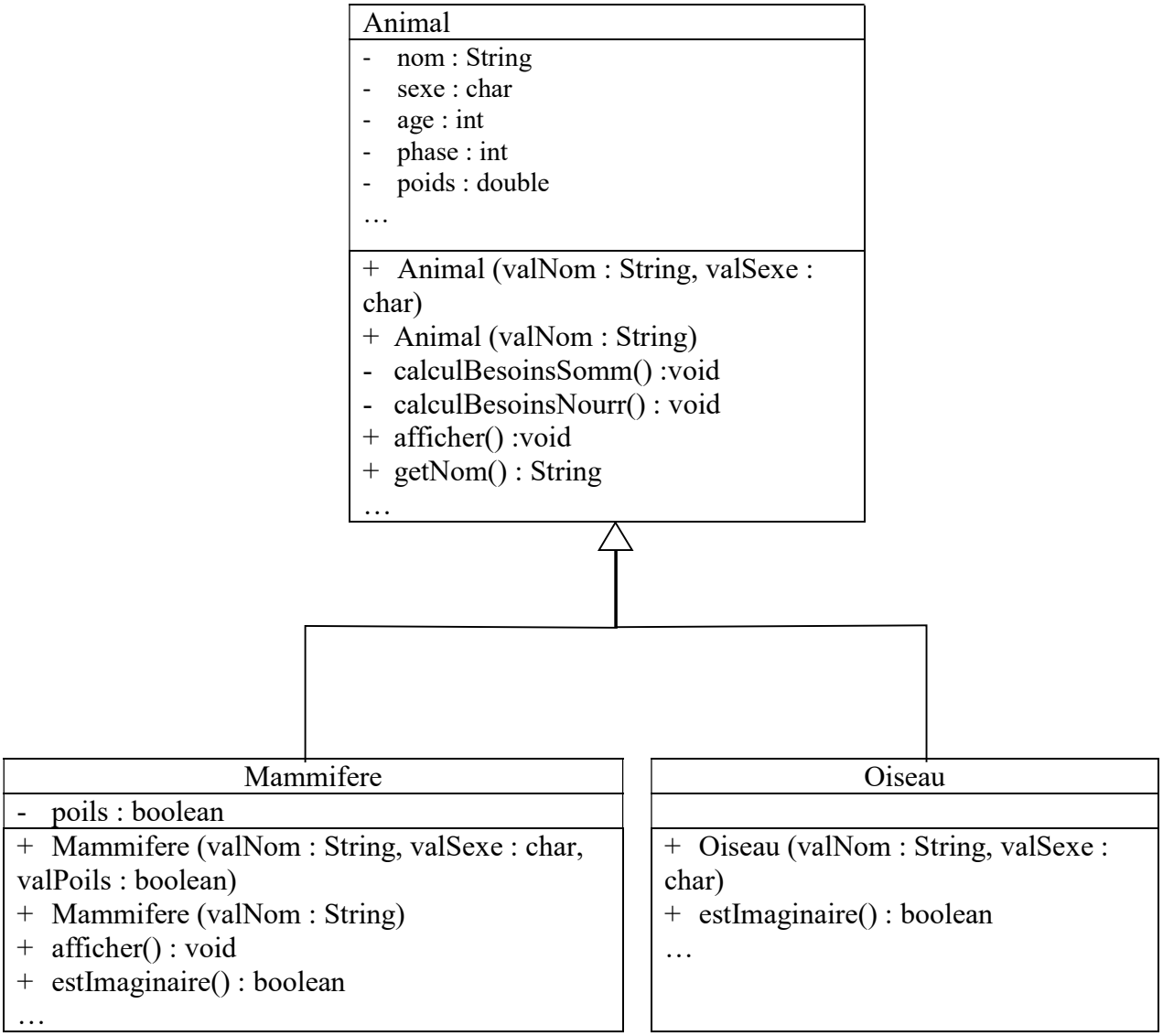
nomObjet = new NomClasse(...) ;

nomObjet.nomAttribut (dans la classe uniquement)

nomObjet.nomMethode (...) ;

L'exemple

La hiérarchie de classes



Les objets