CLASSES ET HERITAGE

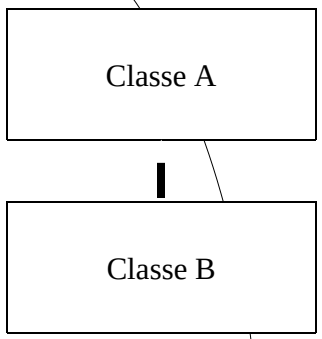
Ludovic Liétard

L'Héritage

- Un des principes fondamentaux de l'approche objet
- Met en relation deux classes A et B
- L'héritage est une relation de spécialisation/généralisation entre A et B

L'Héritage

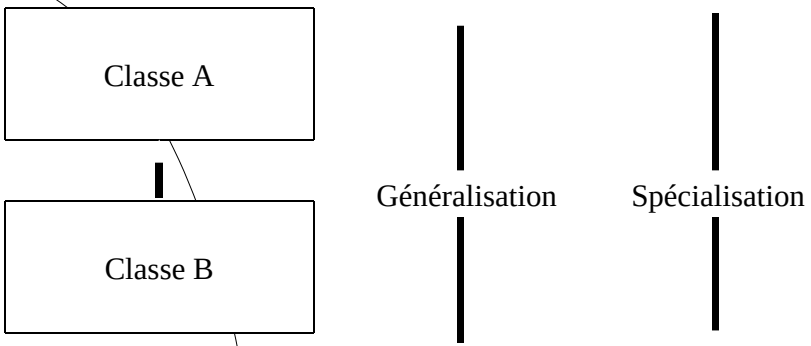
- Pour exprimer qu'une instance de B est « une sorte » d'instance de A



B hérite de A
A est la superclasse de B
B est une sous-classe de A
B étend A

- Une instance de la sous-classe est « une sorte » d'instance de la superclasse

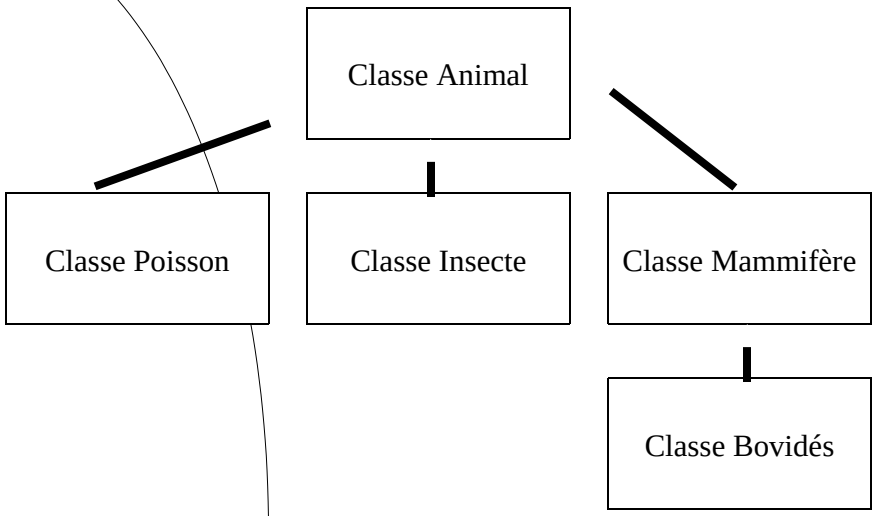
L'Héritage



- La superclasse est une généralisation de la sous-classe
- La sous-classe est une spécialisation de la superclasse

L'Héritage

- On obtient un arbre d'héritage :



L'Héritage

- Une classe peut avoir plusieurs sous-classes
- Une sous-classe peut avoir plusieurs superclasses immédiates (cas de l'héritage multiple)
- En java, une classe n'a qu'une superclasse immédiate (pas d'héritage multiple)
- Par défaut, une classe hérite de la classe **`Object`**

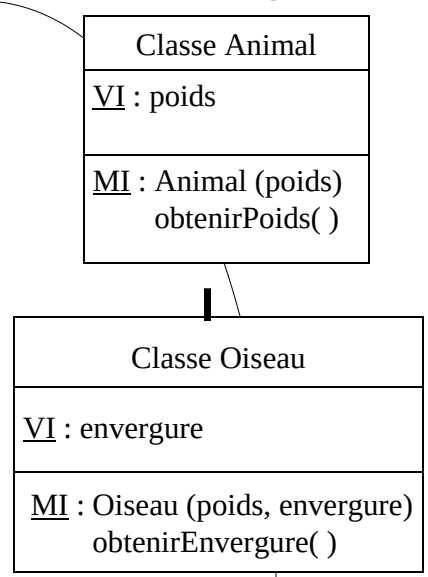
L'Héritage

- Si B hérite de A, la classe B dispose (en plus de ses propres caractéristiques) :

de toutes les variables (instance et classe) de la classe A

de toutes les méthodes (instance et classe) de la classe A

L'Héritage



Une instance d'oiseau aura un **`poids`** et une **`envergure`**.
On pourra lui envoyer les messages **`obtenirPoids()`** et **`obtenirEnvergure()`**.

L'Héritage

- En Java, l'héritage s'exprime par : **`extends`**
- Exemple :

```
class Animal{
        private int poids;

        public Animal(int p){
                poids=p;
         }
         public int obtenirPoids(){
             return poids;
            }
}
```

L'Héritage

```
class Oiseau extends Animal{
        private int envergure;

        public Oiseau(int p, int e){
                super(p);
                envergure = e;
         }
         public int obtenirEnvergure(){
             return envergure;
            }
}
```

L'Héritage

- Chaînage des constructeurs (dans un constructeur):

Un appel à un constructeur de la superclasse se fait à l'aide du mot réservé **`super`** (toujours en première ligne)

Un appel à un constructeur de la même classe se fait à l'aide du mot reservé **`this`** (toujours en première ligne)

Le compilateur se base sur les arguments pour trouver la bonne méthode

L'Héritage

- Chaînage des constructeurs :

```
class Animal{
        private int poids;
        public Animal(int p){
                poids = p;
         }
         public Animal () {
            this(0);
         }
         …
}
```

L'Héritage

- Chaînage des constructeurs :

```
class Oiseau extends Animal{
        private int envergure;
        public Oiseau(int p, int e){
                super(p);
                envergure = e;
        }
        …
}
```

Redéfinition

- Redéfinir une méthode d'instance (overriding):

Lorsqu'une classe B hérite d'une classe A, elle peut redéfinir des méthodes d'instance de la classe A.

Cela signifie que B peut comporter des méthodes d'instance ayant même nom, même type de paramètres et de type de retour que des MI de sa superclasse.

Redéfinition

- Redéfinir une méthode d'instance:

```
class A {
        public void aff(){
                System.out.println(" de classe a ");
        }
}
class B extends A {
        public void aff(){
                System.out.println(" de classe b ");
        }
}
```

Redéfinition

```
class C extends B {
}
```

- Que se passe-t-il lors de l'exécution de :

```
A a=new A();
B b=new B();
C c=new C();

a.aff();
b.aff();
c.aff();
```

Redéfinition

- Ne pas confondre redéfinition (overriding) et surcharge (overloading)
- La redéfinition : dans une sous-classe, même signature que dans la super-classe
- La surcharge : signatures différentes

Redéfinition avec réutilisation

- On peut redéfinir une méthode et réutiliser le code de la méthode héritée

```
class Personne {
      private String nom;
      private String prenom;

     public void aff(){
          System.out.println(nom+ "  " +prenom);
      }
}
```

Redéfinition avec réutilisation

```
class Etudiant extends Personne {
      private int num_groupe;

      public void aff(){
          super.aff();
          System.out.println(num_groupe);
      }
}
```

- Redéfinit l'affichage en réutilisant l'affichage des instances de `Personne`

Réutilisation

- D'une manière générale, le mot-clé super permet de réutiliser le code d'une méthode définie dans la super-classe :

```
super.nom_methode(liste
                  d'arguments);
```

- Cela n'implique pas obligatoirement une redéfinition

Réutilisation

```
class A {
        void faire(){
                System.out.println("Ici en A");
        }
}
class B extends A {
        void faire(){
                System.out.println("Ici en B");
                super.faire();
        }
        void autre(){
                super.faire();
                this.faire();
        }
}
```

Réutilisation

- Que se passe-t-il lors de l'exécution de :

```
B b;
b = new B();
b.autre();
```

- Remarque : l'appel

  `super.super.methode();`

  est interdit

Masquage d'attributs

- Un attribut d'une classe B masque un attribut de sa super classe A si les deux attributs portent le même nom.
- La nouvelle définition masque la définition héritée.

Masquage d'attributs

```
class A {
        private int n;
        ……
}

class B extends A {
        private char n;
        ……
}
```

- Ce phénomène porte également le nom de redéfinition des attributs (shadowed variables)

Conversion (coercition) de classe

- Soit B une sous-classe de A et :

```
A a;
B b;
b = new B();
a = b;
```

- La dernière affectation est légale car une instance de B est « une sorte de A ».

Conversion de classe

- Soit B une sous-classe de A et :

```
A a;
B b,c;
b = new B();
a = b;
c = a; // pose pb
```

- La dernière affectation est possible si transtypage

Conversion de classe

- Transtypage:

```
A a;
B b,c;
// …sous réserve que a soit une instance de B…..
c = (B) a;
```

Deux variantes de la spécialisation

- L'héritage est une relation de spécialisation/généralisation entre deux classes
- La spécialisation peut être :

  a) par enrichissement

  b) par restriction

Deux variantes de la spécialisation

- Spécialisation par enrichissement :

  personne/étudiants/thésard
  animal/oiseau/rapace

- La spécialisation par restriction :

  rectangle/carré

Deux variantes de la spécialisation

- La spécialisation par enrichissement se prête à l'héritage (sans problèmes).

- La spécialisation par restriction ne se prête pas à l'héritage (interdit).