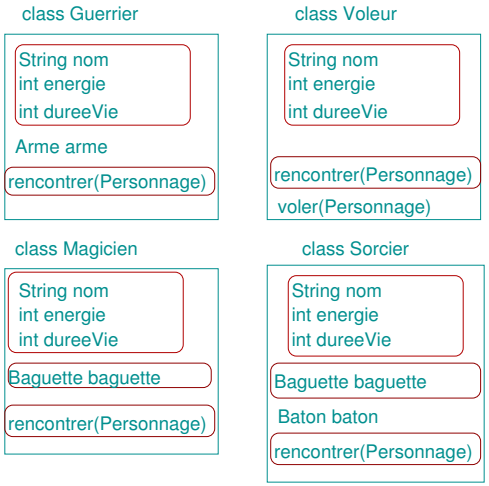
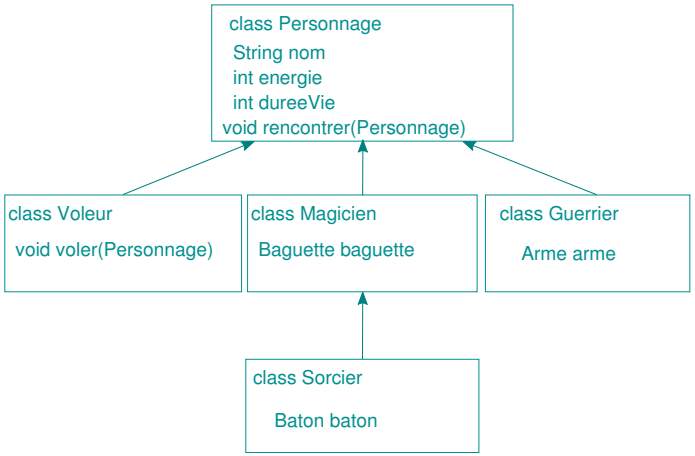
Un exemple

Oublions un peu les rectangles ...

Exemple : classes pour les personnages



Exemple : héritage

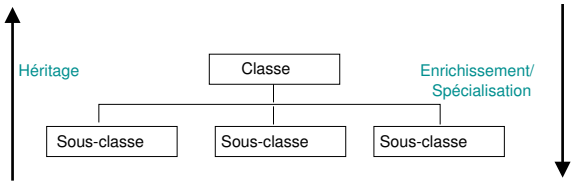


Héritage

Après les notions d'*encapsulation* et d'*abstraction*, le troisième aspect essentiel de la « Programmation Orientée Objet » est la notion d'**héritage**.

L'héritage représente la relation «**est-un**».

Il permet de créer des classes *plus spécialisées*, appelées **sous-classes**, à partir de classes plus générales déjà existantes, appelées **super-classes**.



Héritage (2)

Lorsqu'une sous-classe `C1` est créée à partir d'une super-classe `C`,

- le type est *hérité* : un `C1` **est** (aussi) **un** `C`
- `C1` va *hériter* de l'ensemble :
  - des attributs de `C`
  - des méthodes de `C`
    (sauf les constructeurs)

  ☞ Les attributs et méthodes de `C` vont être disponibles pour `C1` sans que l'on ait besoin de les redéfinir explicitement dans `C1`.
- Par ailleurs :
  - des attributs et/ou méthodes supplémentaires peuvent être définis par la sous-classe `C1`
    ☞ **enrichissement**
  - des méthodes héritées de `C` peuvent être redéfinies dans `C1`
    ☞ **spécialisation**

Héritage : exemple

Lorsqu'une sous-classe `C1` (ici `Guerrier` ou `Voleur`) est créée à partir d'une super-classe `C` (ici `Personnage`),

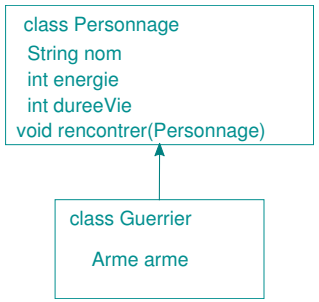
- le type est *hérité* : un `Guerrier` **est** (aussi) **un** `Personnage` :

```
Personnage p;
Guerrier g;
// ...
p = g;
// ...
void afficher(Personnage);
// ...
afficher(g);
```

Héritage : exemple

Lorsqu'une sous-classe `C1` (ici `Guerrier` ou `Voleur`) est créée à partir d'une super-classe `C` (ici `Personnage`),

- `Guerrier` va *hériter* de l'ensemble des attributs et des méthodes de `Personnage`
  (sauf les constructeurs)




```
Guerrier g = new Guerrier(...);
Voleur   v = new Voleur(...);

g.rencontrer(v);
//...
// dans une méthode de Guerrier
   energie = //...
```

Héritage : exemple

Lorsqu'une sous-classe `C1` (ici `Guerrier` ou `Voleur`) est créée à partir d'une super-classe `C` (ici `Personnage`),

- des attributs et/ou méthodes supplémentaires peuvent être définis par la sous-classe `Guerrier` : `arme`
- des méthodes héritées de `Personnage` peuvent être redéfinies dans `Voleur` : `rencontrer(Personnage)`

Héritage (3)

L'héritage permet donc :

- d'*expliciter des relations structurelles et sémantiques* entre classes
- de *réduire les redondances* de description et de stockage des propriétés

Attention !

- l'héritage doit être utilisé pour décrire une relation **« est-un »** ("is-a")
- il ne doit **jamais** décrire une relation « a-un »/« possède-un » ("has-a")

Transitivité de l'héritage

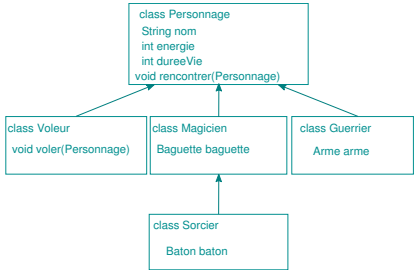
Par transitivité, les instances d'une sous-classe possèdent :

- les attributs et méthodes (hors constructeurs) de l'ensemble des classes parentes (super-classe, super-super-classe, etc.)

Enrichissement par héritage :

- crée un *réseau de dépendances* entre classes,
- ce réseau est organisé en une *structure arborescente* où chacun des nœuds hérite des propriétés de l'ensemble des nœuds du chemin remontant jusqu'à la racine.

☞ ce réseau de dépendances définit une **hiérarchie de classes**



Sous-classe, Super-classes

Une super-classe :

- est une classe « parente »
- déclare les attributs/méthodes communs
- peut avoir plusieurs sous-classes

Une sous-classe est :

- une classe « enfant »
- étend **une seule** super-classe
- hérite des attributs, des méthodes et du type de la super-classe

Un attribut/une méthode hérité(e) peut s'utiliser comme si il/elle était déclaré(e) dans la sous-classe au lieu de la super-classe (en fonction des droits d'accès, voir plus loin)

☞ On évite ainsi la duplication de code

Passons à la pratique...

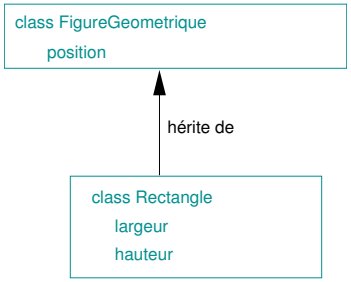
Définition d'une sous-classe en Java :

Syntaxe :

```
class NomSousClasse extends NomSuperClasse
{
 /* Déclaration des attributs et méthodes
    spécifiques à la sous-classe */
}
```

Exemple :

```
class Rectangle extends FigureGeometrique
{
   private double largeur;
   private double hauteur;
   // ...
}
```



Pratique : exemple 2

```
class Personnage {
  // ...
}
// ...
class Guerrier extends Personnage {
  private Arme arme;
  // constructeurs, etc.
}
```