MOOC semaine 4
Héritage

Message1

Héritage == ++Ré-utilisation ;

Message2

Clef du succès: distinguer **“Est un(e)”** de **“Possède un(e)”**

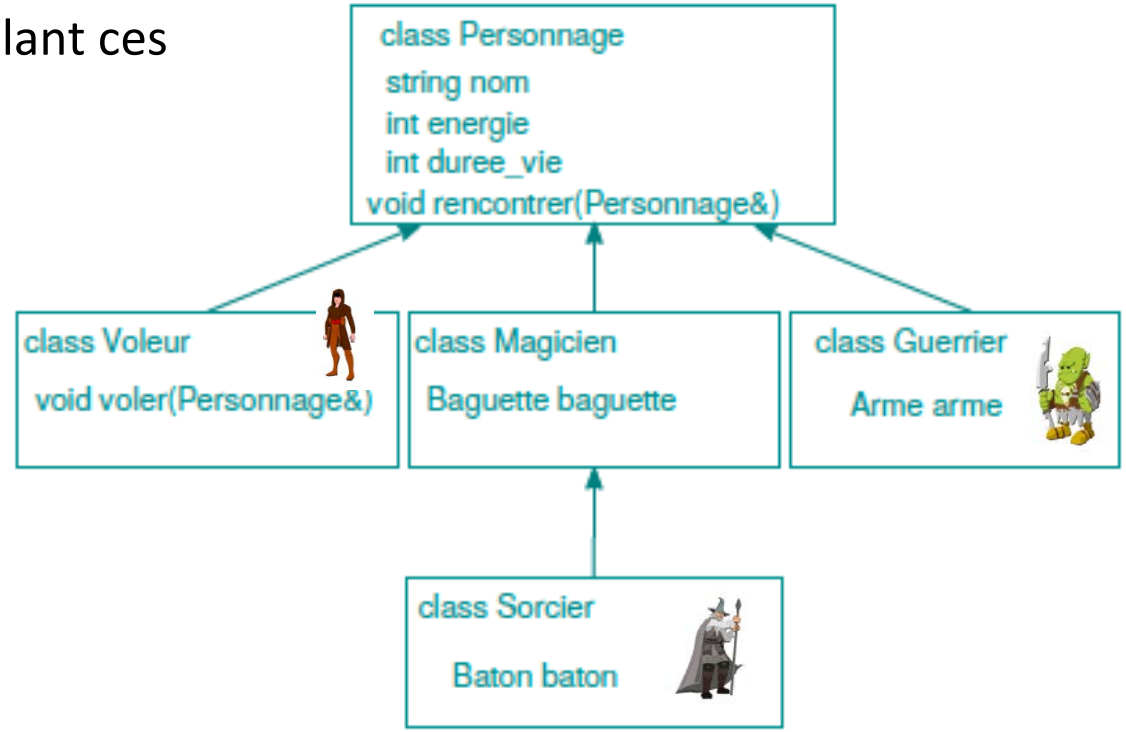
Plan:

- Principe et motivations du mécanisme d’héritage
- Rappel de l’exemple du MOOC
- Distinction entre “Est un(e)” et “Possède un(e)”

Héritage == ++ ré-utilisation

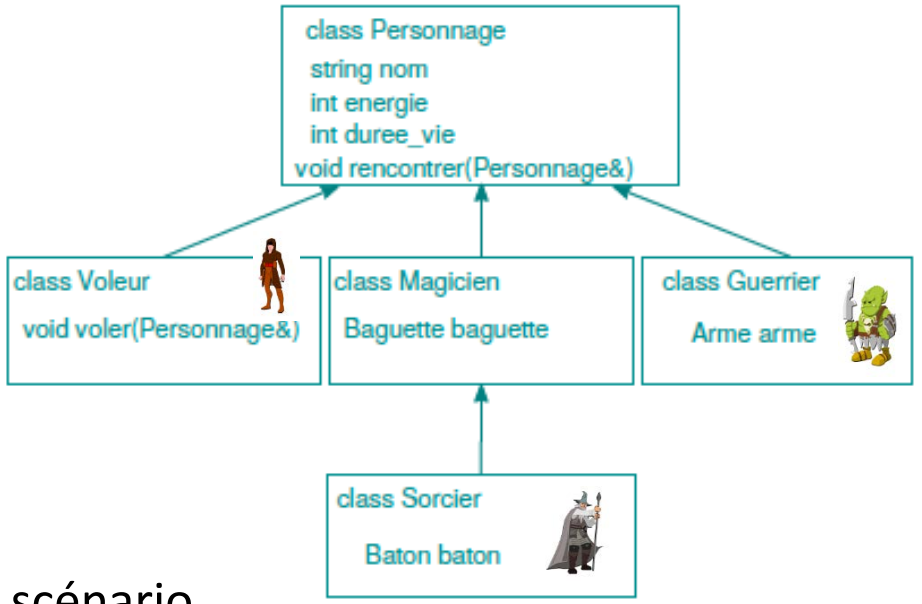
Motivation: lorsqu'un ensemble d'entité présente un forte cohérence sémantique

- Partager un sous-ensemble de propriétés communes
- Eviter de dupliquer des méthodes manipulant ces propriétés communes
- Offrir une base de code cohérente et validée pouvant être **étendue** par **l'ajout de sous-classes**
- Fonctionnement **robuste** et **fiable** d'une base de code validée quand un tiers ajoute ses sous-classes.



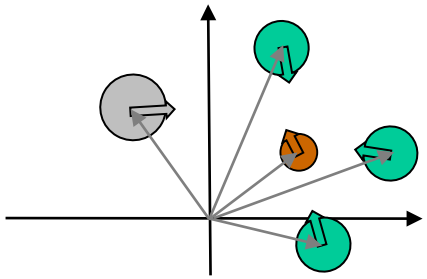
Distinguer “Est un(e)” de “Possède un(e)”

La relation « Possède un(e) » est la relation déjà connue entre une instance d’une classe et ses attributs :
=> Un Personnage possède : nom, energie, duree_vie



La relation « Est un(e) » est celle qui existe entre une sous-classe et sa classe parente:
⇒ **Un Sorcier est un Magicien**
⇒ **Un Magicien est un Personnage**

Problème: supposons que pour les besoins d’une simulation de scénario de film, je veux pouvoir éviter des collisions et calculer des stratégies de déplacement. Pour cela l’information de **position**, **orientation** et de **rayon d’espace occupé** des personnages est nécessaire.



Questions: Faut-il créer des sous-classes ? Ajouter l’information à chaque sous-classe existante ? Créer une nouvelle superclasse dont Personnage va hériter ?

Avant de vous lancer avec l'héritage, il faut :

- **Savoir *utiliser* une hiérarchie de classes (ex: GTKmm)**

- Maîtriser le **principe de ré-utilisation** sans héritage
  - Identifier les limitations de cette approche (gestion de la complexité)

- Maîtriser l'analyse des relations entre les éléments d'un modèle
  - pour distinguer deux types de relations (diagramme de classes):
    - les relations **« Possède un(e)»**
      => conception classique = attribut d'une classe ou struct

    - Des relations **« Est un(e) »**
      => conception d'une hiérarchie de classes

- Guide: Principe de Parcimonie