ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

02
Chapitre

Un paradigme : La Conception Orientée Objet

1I2AC1 : Génie logiciel et Conception orientée objet

Régis Clouard, ENSICAEN - GREYC

« N'importe quel programmeur peut écrire
du code que l'ordinateur comprend.
Les bons programmeurs écrivent du code
que les humains peuvent comprendre. »
Martin Fowler

Plan du chapitre

1
Le paradigme objet

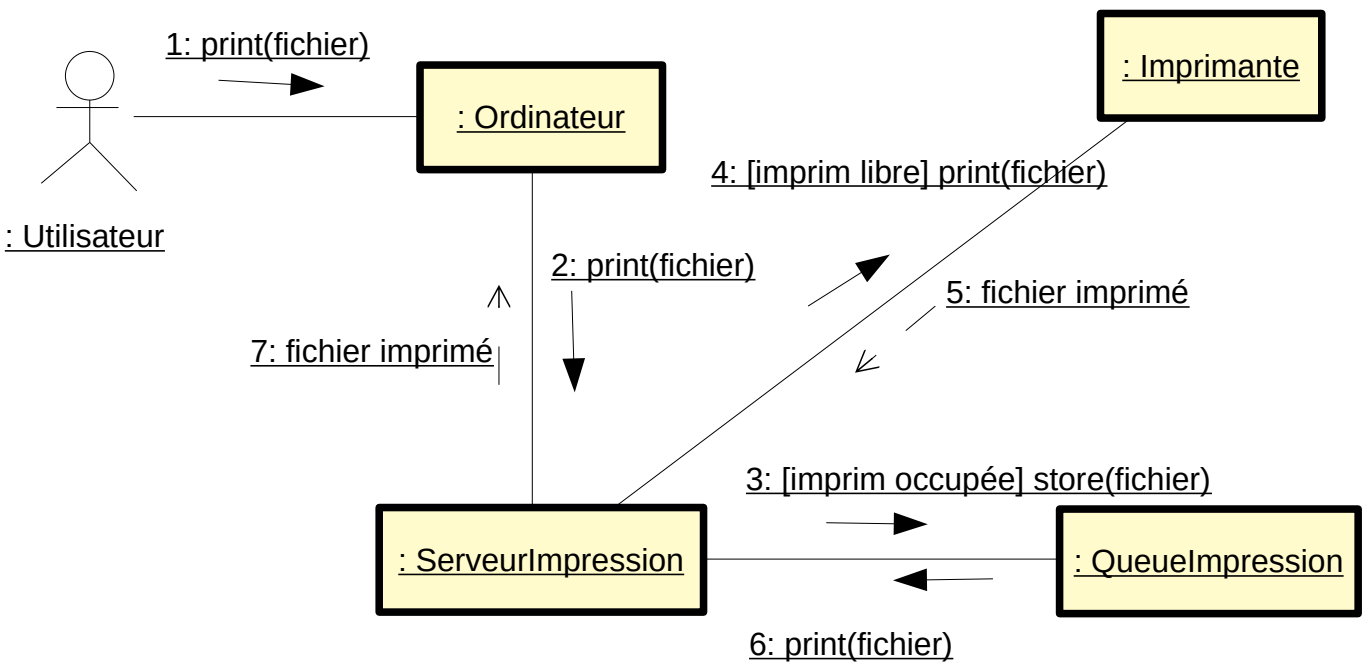
2
Les objets et le principe d'encapsulation)

3
Les classes

4
Associations entre classes

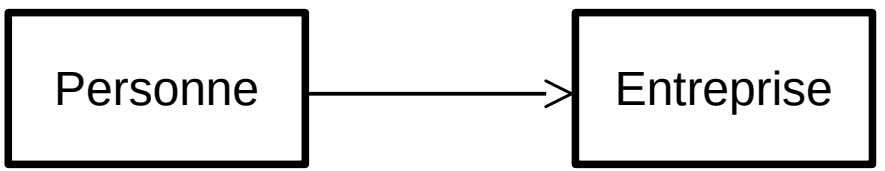
Association

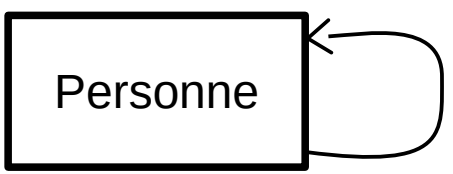
- Un objet ne doit pas être omniscient mais au contraire spécialisé
  - Sinon cela revient à faire de la conception procédurale
- Il doit donc faire appel aux services d'autres objets qu'il connaît par la liste de ses associations



Association

- Pour qu'un objet puisse utiliser les services d'un autre objet, il faut qu'il connaisse son emplacement mémoire
  - C'est le rôle des associations
- Association
  - Définie par la classe (réserve un pointeur en mémoire pour les objets)
  - Instanciée par les objets de la classe (met la valeur du pointeur)





Code Java

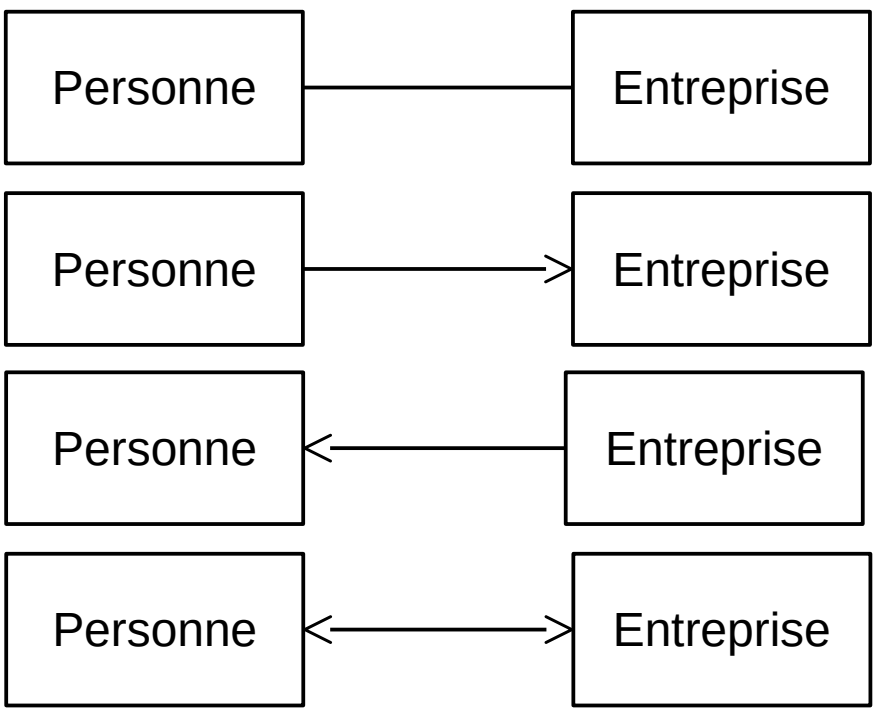
- En Java, les associations sont implémentées par des données membres
  - **Mais les associations ne sont pas des attributs**

Décoration des associations

- Intention de la décoration
  - documenter l'association
  - donner des directives d'implémentation
- 3 types de décoration
  a) Navigabilité
  b) Rôle
  c) Multiplicité

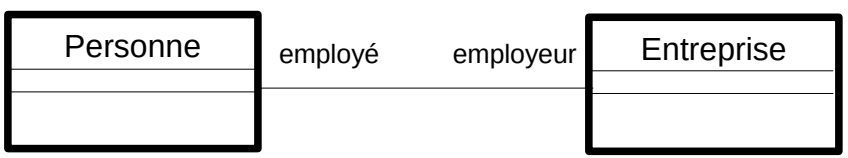
a) Navigabilité

- Navigabilité: sens de l'association
  - Notation : flèche au bout du trait



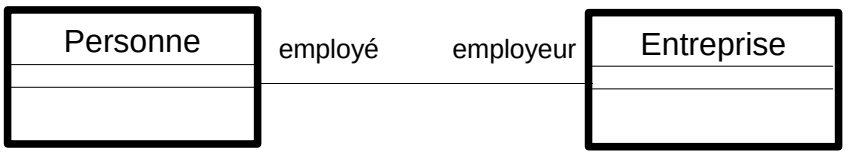
b) Rôle

- Rôle : sémantique de l'association
  - À chaque extrémité de l'association



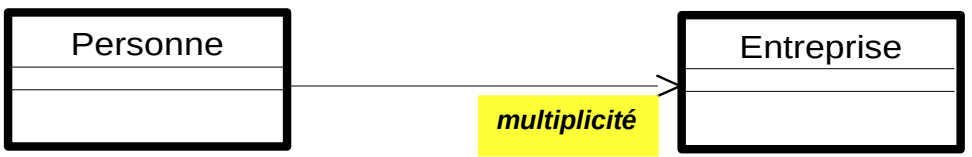
Code Java

- En Java, le rôle donne le nom de l'association



c) Multiplicité

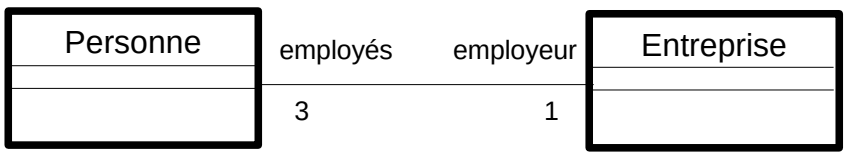
- Cardinalité à chaque extrémité de la navigabilité



1	Un et un seul objet dans l'association (par défaut)
0..1	Zéro ou un objet
M..N	De M à N objets
*	De zéro à plusieurs objets
1..*	De 1 à plusieurs objets
N	Exactement N objets

Code Java

- En Java, la multiplicité est implémentée par
  - une donnée membre (≤ 1)
  - un tableau de données membres ou une liste de données membres (> 1)



Importance de la multiplicité

Exemple

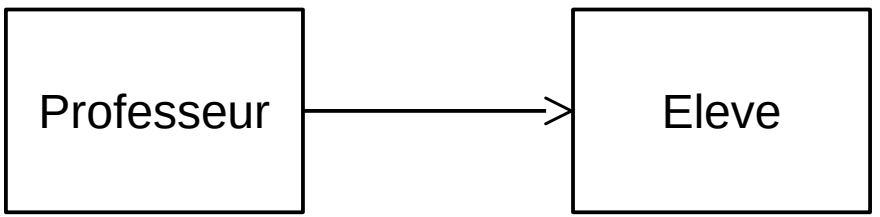
Relation de mariage

Typage d’association

- Intention du typage
  - Ajouter de la sémantique à la modélisation
  - Donner des directives d’implémentation
- 3 types d’association
  a) Standard
  b) Agrégation
  c) Composition

(a) Association : standard

- Sémantique : **connaît** (pour utiliser les services)
- Exemple : professeur - élève

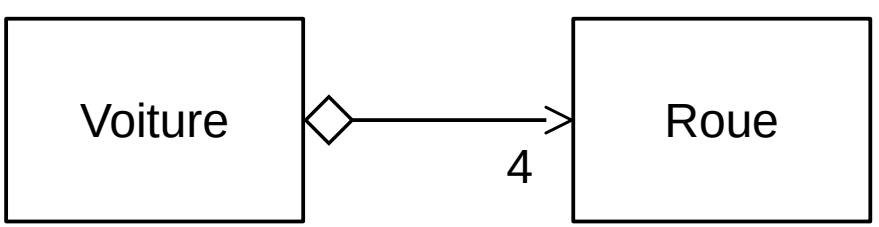


(b) Association : agrégation

- Sémantique : **possède** (relation ensembliste)
- Exemples :

Relation	Exemple
Composé / Composant	*Voiture / Roues*
Collection / Élément	*Forêt / Arbres*
Espace / Position	*Désert / Oasis*
Événement / Étape	*Document / Chapitre*

- Notation : losange creux



Agrégation en Java

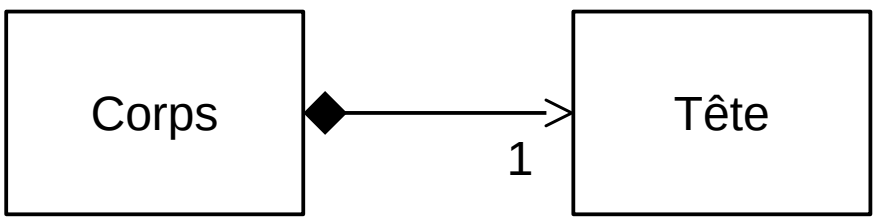
- Implication en Java
  - Il faut ajouter les deux opérations suivantes dans la classe agrégat :
    - `add(element)`, `remove(element)`

(c) Association : composition

- Sémantique : **est constitué de** (relation compositionnelle)
  - L'objet composite a la responsabilité de l'existence et du stockage de l'objet composé

Relation	Exemple
Corps / Portion	*Corps / Tête*
Matière / Substance	*Eau / Hydrogène*
Activité / Phase	*Achat / Paiement*

  - Notation : losange plein



Composition en Java

- Conséquence sur le code
  - Ajouter du code dans la classe Composite qui crée et détruit les objets de la classe Composant

Quiz

- Éleveur → Cheval
- Joker → Cheval
- Cheval → Tête
- Cheval → Cœur
- Cheval → Selle
- École → Étudiant
- Carte mère → microprocesseur
- GAB → Billet

Plan du chapitre

1
Le paradigme objet

2
Les objets et le principe d'encapsulation)

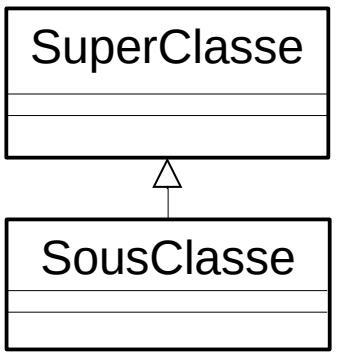
3
Les classes

4
Associations entre classes

5
Héritage et polymorphisme

(2) Héritage

- Relation de subsomption
  - Une classe hérite de tous les éléments d'un superclasse

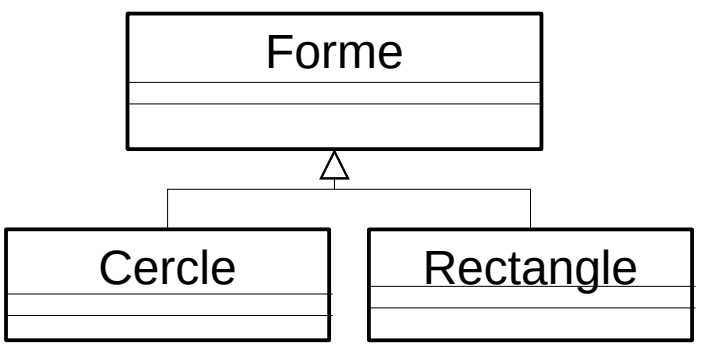


Héritage

- Intention 1 : **généralisation**
- Exemple : Formes géométriques
  - Toutes les formes possèdent une dimension et une position **avec la même sémantique**
  - Toutes les formes possèdent une méthode `deplace()` **avec la même sémantique**
  - → On peut les factoriser dans une super-classe Forme

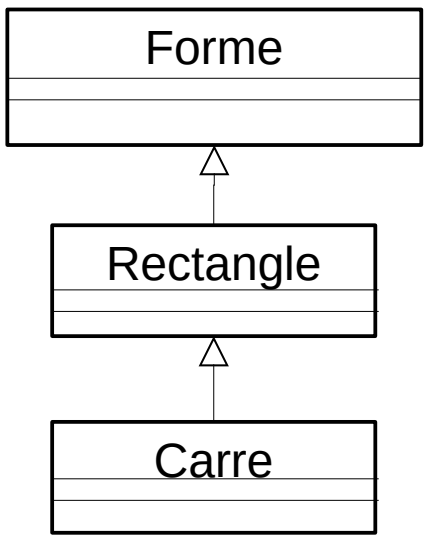
Rectangle
dimension = 2
position=(0,0)
void deplace()

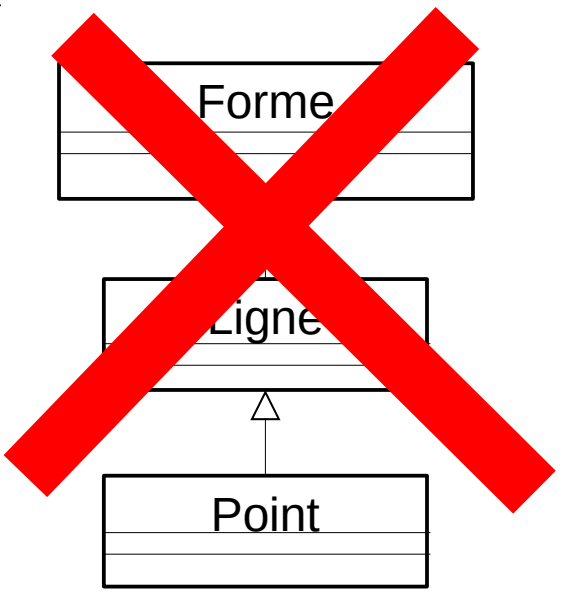
Cercle
dimension = 2
position = (10,30)
void deplace()



Héritage

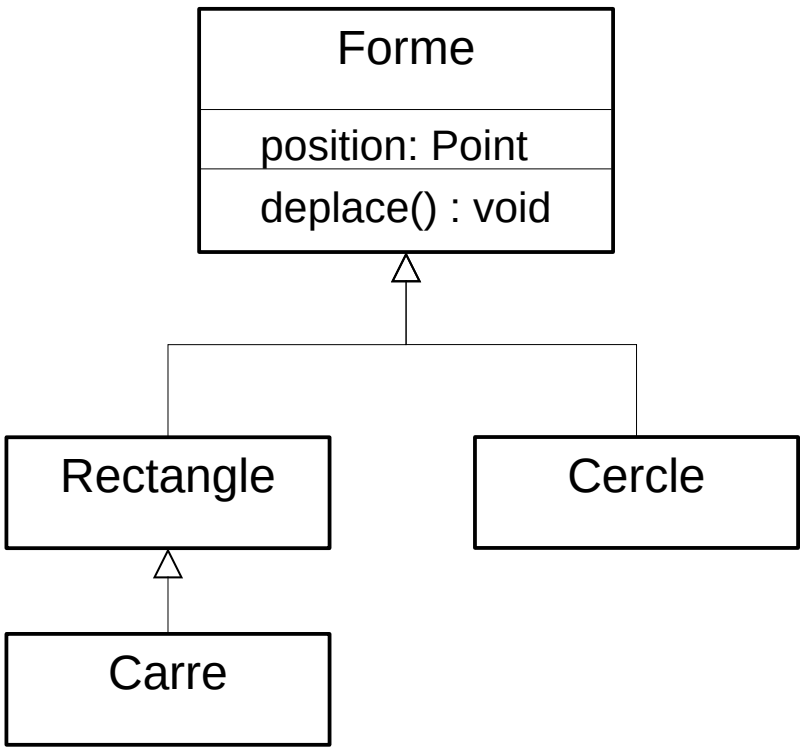
- Intention 2 : **spécialisation**
- Exemple :
  - Un carré est un cas particulier de rectangle





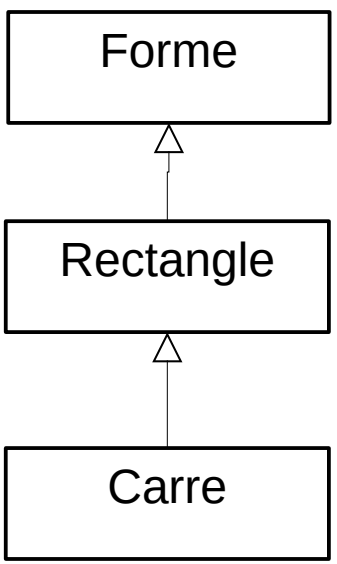
Héritage en Java

- L'héritage est implémenté par le mot clé **extends**



Transtypage

- **Upcasting** : surclassement
- **Downcasting** : sous-classement



Quiz transtypage

- Forme c1 = new Carre(); √
- Rectangle c2 = new Carre(); √
- Carre c3 = new Rectangle(); X
- Forme r = new Rectangle(); √
- Rectangle r1 = (Rectangle)r; √
- Carre c = (Carre)r; X