



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

25

1

Le paradigme
objet

2

Les objets et le
principe
d'encapsulation)

3

Les classes

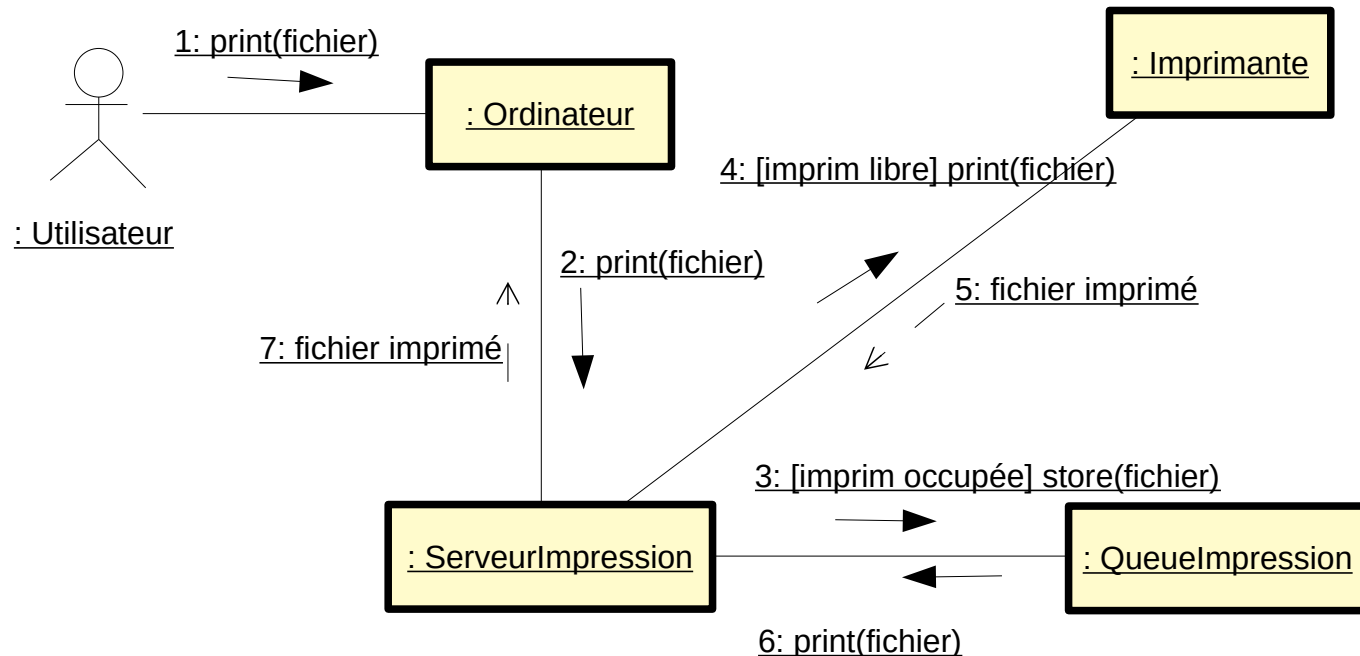
4

Associations
entre
classes

Relation entre classes

26

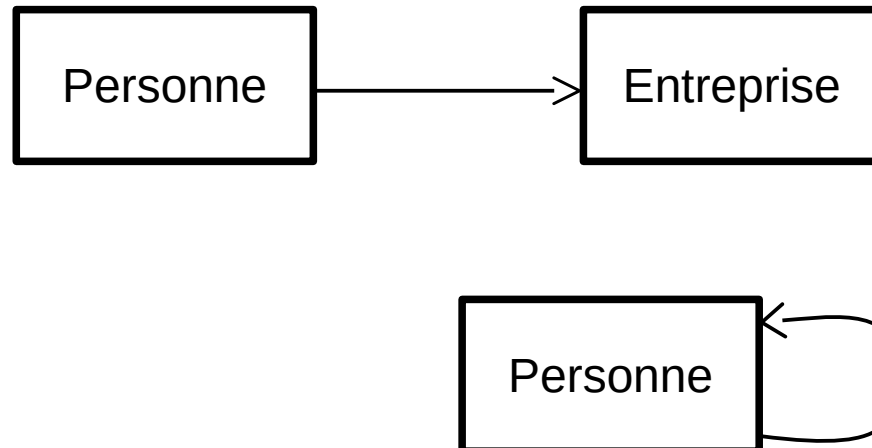
- Un objet ne doit pas être omniscient mais au contraire spécialisé
 - Sinon cela revient à faire de la conception procédurale
- Il doit donc faire appel aux services d'autres objets qu'il connaît par la liste de ses relations



Un 1^{er} type de relation : Association

27

- 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 pour les objets associables)
 - Instanciée par les objets de la classe (met la valeur du pointeur)



Décoration des associations

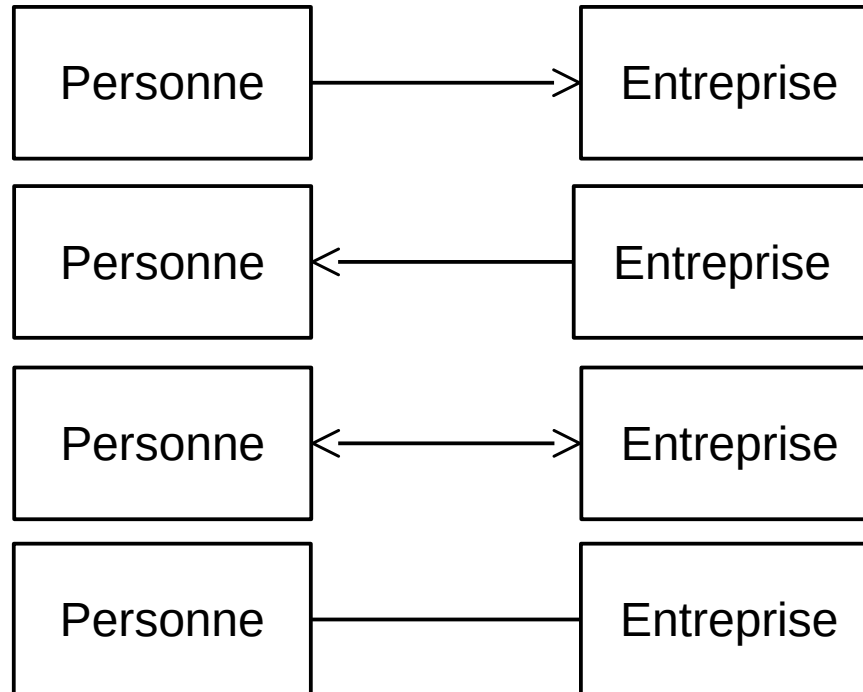
28

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

a) Navigabilité

29

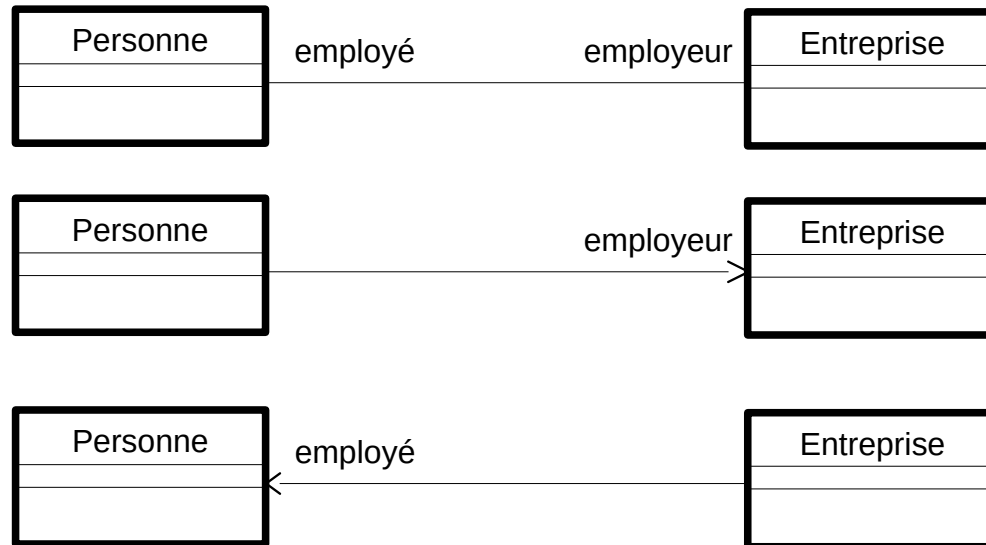
- Navigabilité: sens de l'association
 - Notation : flèche au bout du trait



b) Rôle

30

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



c) Multiplicité

31

- 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

Importance de la multiplicité

32

Exemple

Relation de mariage

Code Java

33

- En Java, les associations sont implémentées par des données membres, **mais les associations ne sont pas des attributs**
- Le rôle donne le nom de l'association
- La multiplicité est implémentée par
 - une donnée membre (≤ 1)
 - un tableau de données membres (> 1)



Typage d'association

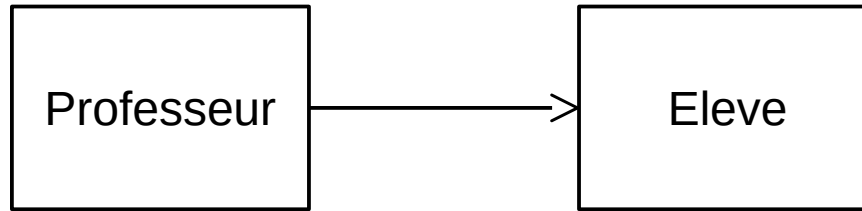
34

- 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

35

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



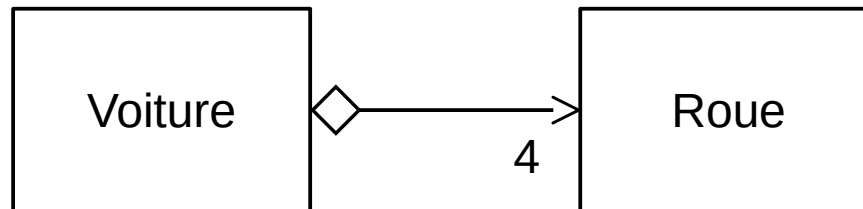
(b) Association : agrégation

36

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

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

- Notation : losange creux



Agrégation en Java

37

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

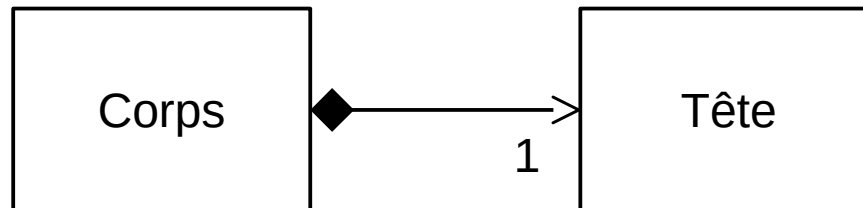
(c) Association : composition

38

- 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	<i>Corps / Tête</i>
Matière / Substance	<i>Eau / Hydrogène</i>
Activité / Phase	<i>Achat / Paiement</i>

- Notation : losange plein



Composition en Java

39

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

Quiz

40

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

Plan du chapitre

41

1
Le paradigme
objet

2
Les objets et le
principe
d'encapsulation)

3
Les classes

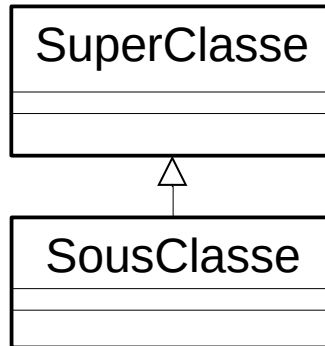
4
Associations
entre
classes

5
Héritage
et
polymorphisme

Un 2^e type de relation : Héritage

42

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



Héritage

43

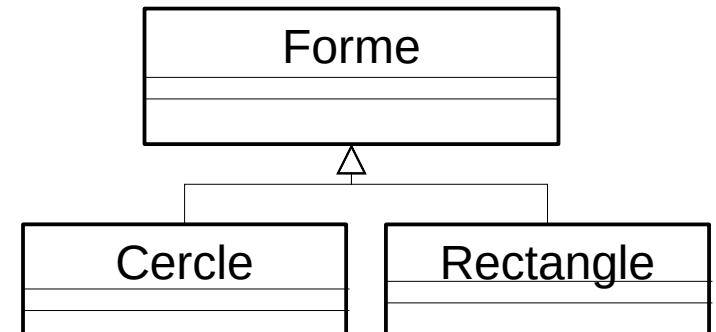
- Intention 1 : **généralisation**
- Exemple : Formes géométriques
 - Toutes les formes possèdent une dimension et une position **avec la même sémantique**
 - Toutes les formes possèdent une méthode `deplace()` **avec la même sémantique**
 - → On peut les factoriser dans une super-classe `Forme`

Rectangle

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

Cercle

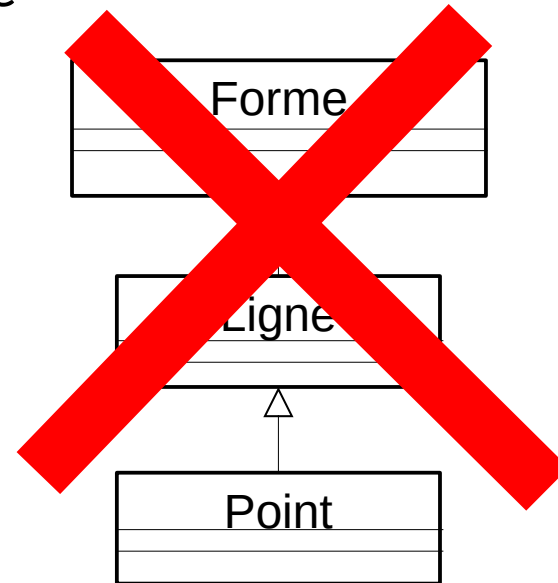
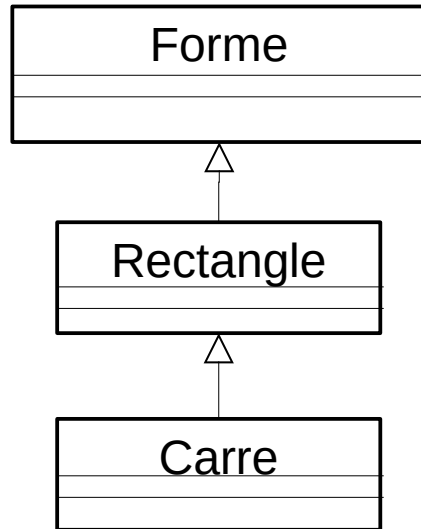
dimension = 2
position = (10,30)
void deplace()



Héritage

44

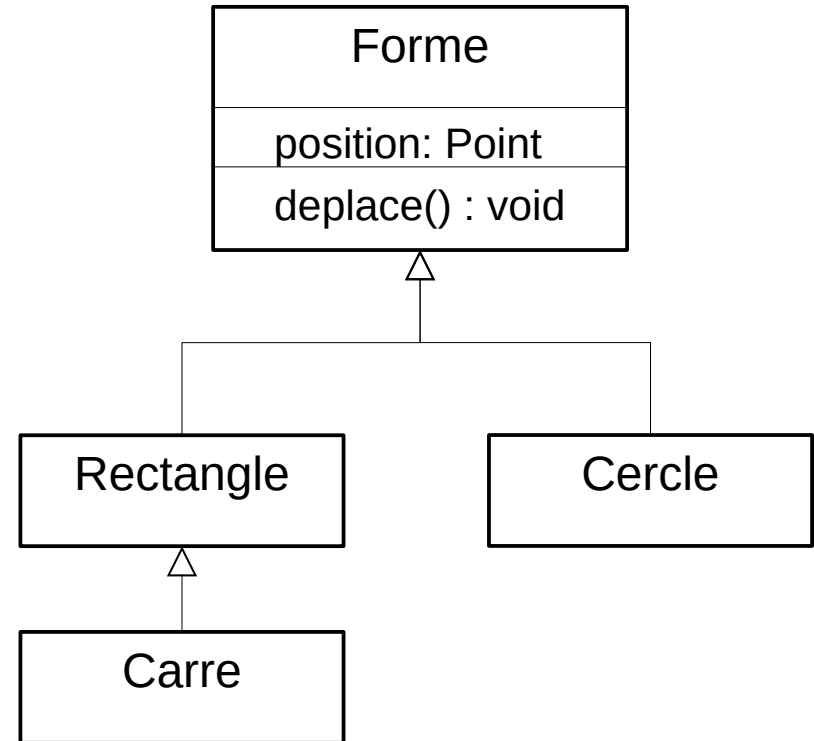
- Intention 2 : **spécialisation**
- Exemple :
 - Un carré est un cas particulier de rectangle



Héritage en Java

45

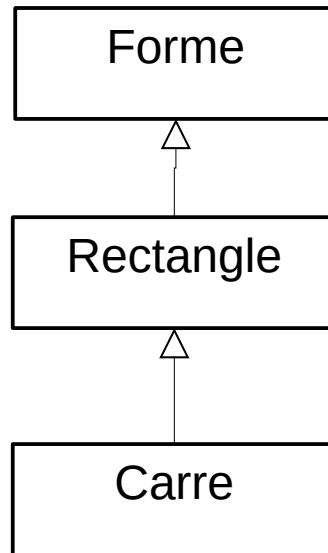
- L'héritage est implémenté par le mot clé **extends**



Transtypage

46

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



Quiz transtypage

47

- `Forme c1 = new Carre();` ✓
- `Rectangle c2 = new Carre();` ✓
- `Carre c3 = new Rectangle();` ✗
- `Forme r = new Rectangle();` ✓
- `Rectangle r1 = (Rectangle)r;` ✓
- `Carre c = (Carre)r;` ✗

