Plan du chapitre

1
Le paradigme objet

2
Les objets

3
Les classes

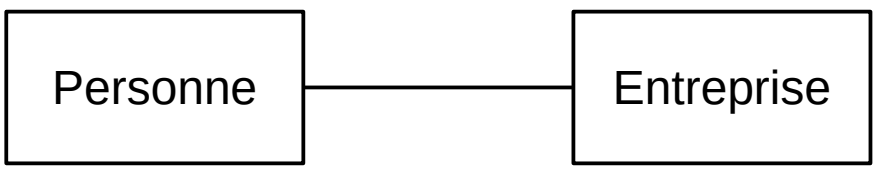
4
Relations entre classes

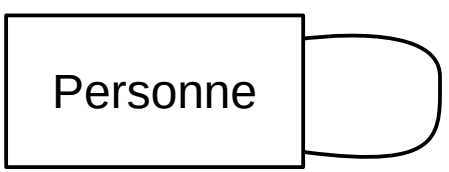
Types de relation

- 3 types de relations entre classes
  1) Association
  2) Dépendance
  3) Héritage

(1) Association

- Pour qu'un objet puisse utiliser les services d'un autre objet, il faut qu'il connaisse son emplacement mémoire
- Association
  - Définit par la classe
  - Instanciée par l'objet





Code Java

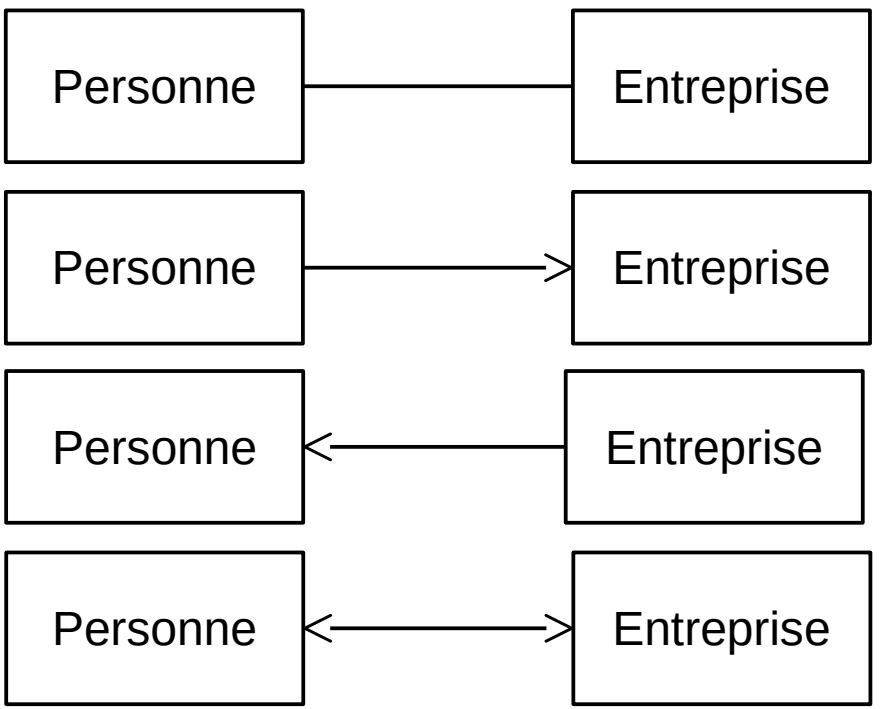

```
class Personne {
   Entreprise entreprise;
}

class Entreprise {
   Personne personne;
}

Personne toto = new Personne();
Entreprise airbus = new Entreprise();
toto.entreprise = airbus;
airbus.personne = toto;
```

Décoration des associations

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

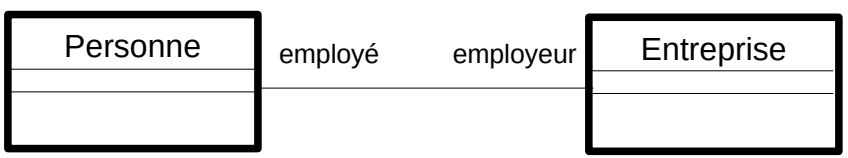
a) Navigabilité

- Navigabilité: sens de l'association



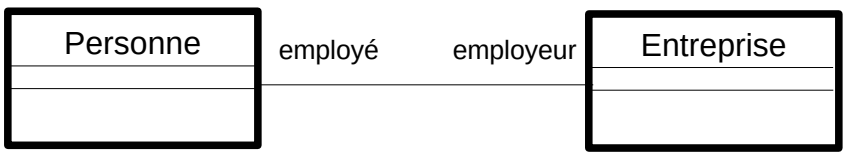
b) Rôle

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



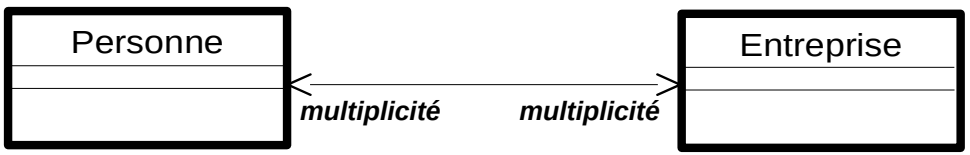
Code Java


```
class Personne {
   Entreprise employeur;
}

class Entreprise {
   Personne employe;
}

Personne toto = new Personne();
Entreprise airbus = new Entreprise();
toto.employeur = airbus;
airbus.employe = toto;
```



c) Multiplicité

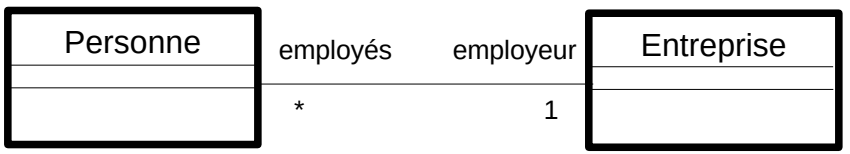
- Cardinalité à chaque extrémité



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

Multiplicité

- Implication dans le code




```
class Personne {
   Entreprise employeur;
}

class Entreprise {
   Personne[] employes;
}

Personne toto = new Personne();
Entreprise airbus = new Entreprise();
toto.employeur = airbus;
airbus.employe[0] = toto;
```

Multiplicité

Exemple

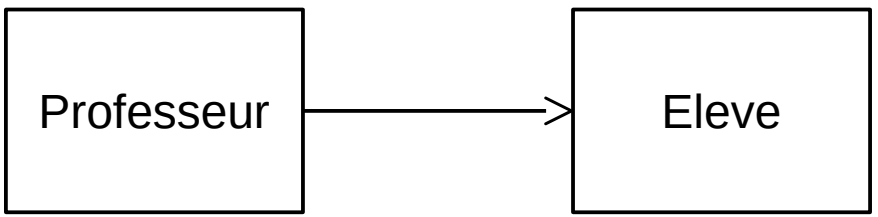
Relation de mariage

Types d’association

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

(a) Association : standard

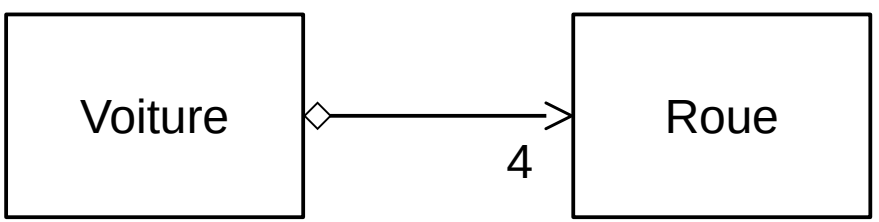
- Sémantique : **connaît** (pour utiliser les services)
- Exemple : professeur - élève



(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*



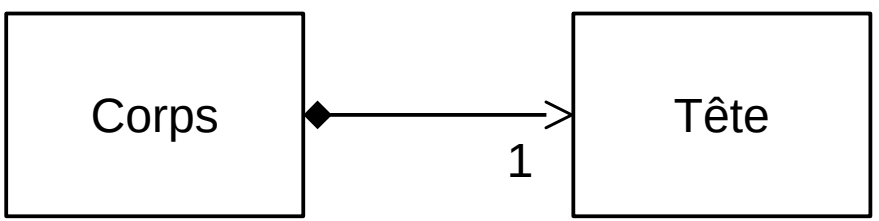
Association : agrégation

- Implication dans sur le code
  - Opération `add()`, `remove()` dans la classe agrégat

(c) Association : composition

- Sémantique : **est constitué de** (relation de 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*



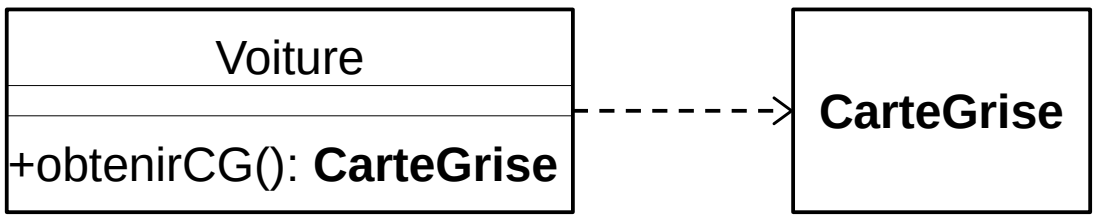
Association : composition

- Conséquence sur le code
  - La classe « Composant » n'est visible que de la classe « Composite »
  - Ajouter du code qui crée et détruit les objets de la classe « Composite »

Quiz

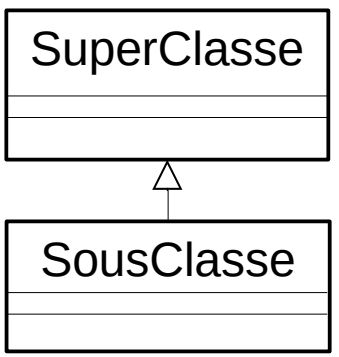
- Joker → Cheval
- Éleveur → Cheval
- Cheval → Tête
- Cheval → Cœur
- Cheval → Selle
- École → Étudiant
- Carte mère → microprocesseur
- GAB → Billet

(2) Dépendance



(3) Héritage

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



Héritage

- Intention 1 : **généralisation**
- Exemple :
  - toutes les formes possèdent une dimension et une position
  - Toutes les formes possèdent une méthode `deplace()`
    → On peut les factoriser dans une super-classe (*maintenance*)

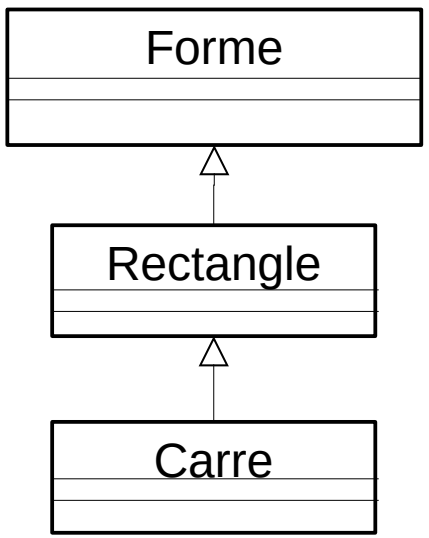
Carré
dimension = 2
position=(0,0)
void deplace()

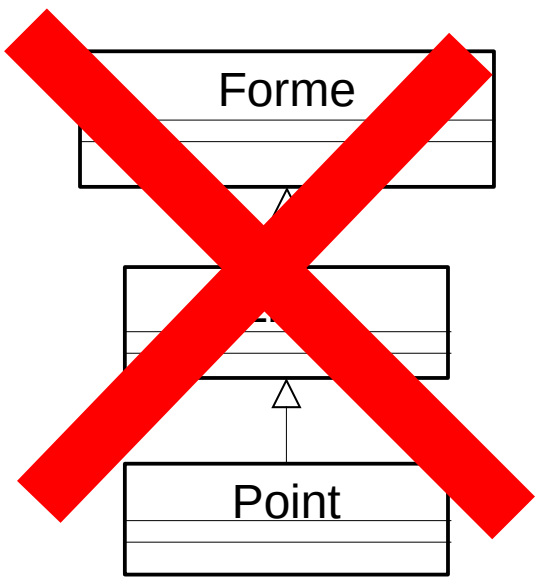
Cercle
dimension = 2
position = (10,30)
void deplace()

Sphere
dimension = 3
position= (100, 20)
void deplace()

Héritage

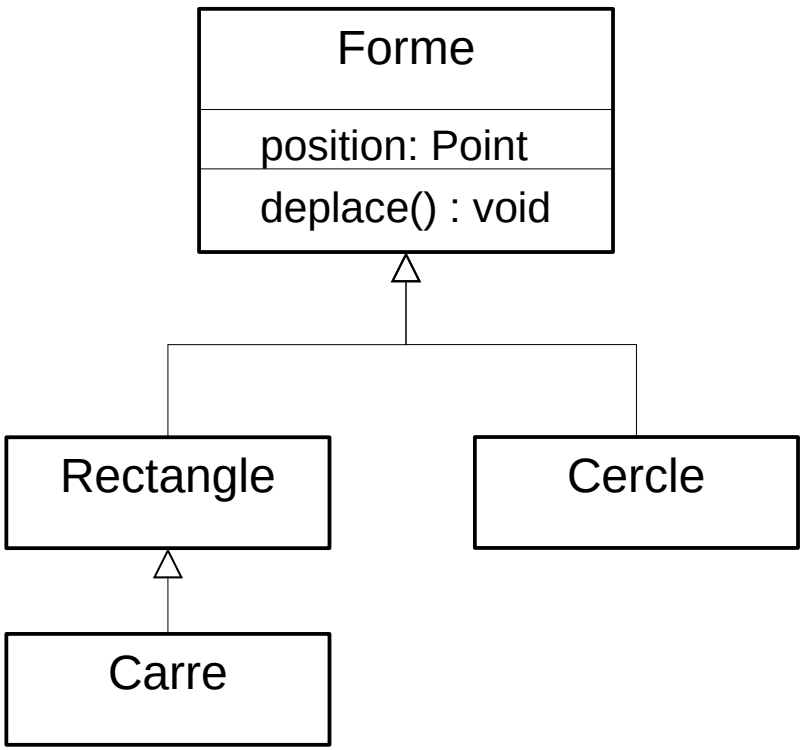
- Intention 2 : **spécialisation**
- Exemple :
  - Un carré est un cas particulier de rectangle





Héritage en Java


```
class Forme {
  Point position;
  Float deplace(Point offset) { ...}
}

class Cercle extends Forme {
}

class Rectangle extends Forme {
}

class Carre extends Rectangle {
}
```



Transtypage

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