TRADUCTION D'UN MODÈLE UML EN JAVA EN MODÈLE RELATIONNEL

UML – Diagramme de Classes

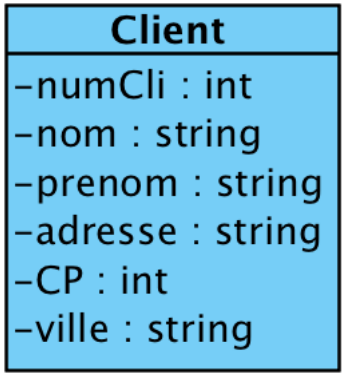
V. Deslandres

IUT - Université Lyon1

DCL → MODÈLE RELATIONNEL

Traduction UML / Relationnel

- Chaque classe **persistante** devient une relation. Chaque attribut de la classe devient un attribut de la relation. L'identifiant de classe devient la clef primaire de la relation.

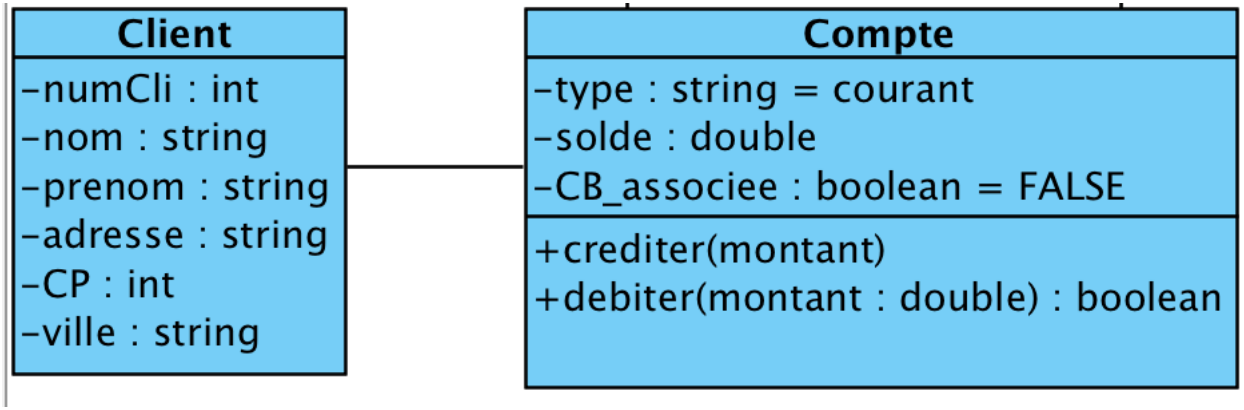


- Ex.:

CLIENT (<u>numCli</u>, nom, prénom, adresse, CP, ville)

Association 1-1

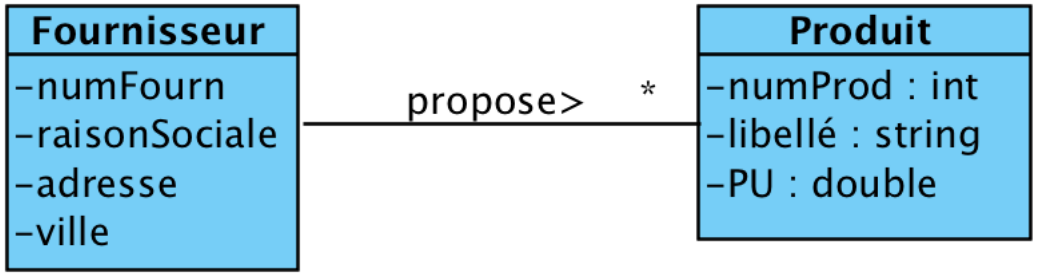
- Traduite par un identifiant placé en clef étrangère dans l'une des relations



- Ex. si un client possède un compte :
  - COMPTE(<u>numCompte</u>, type, solde, CB_associee)
  - CLIENT(<u>numCli</u>, *numCompte,* nom, prénom, adresse, CP, ville)

Association 1-n

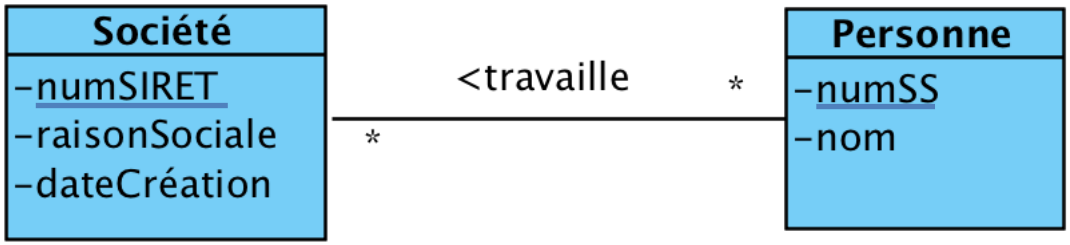
- On place l'identifiant de la relation « 1 » en clef étrangère de la relation « n »



- Ex. si un fournisseur propose différents produits
  - PRODUIT(numProd, libellé, PU, *numFourn*)
  - FOURN(numFourn, raisonSociale, adresse, ville)

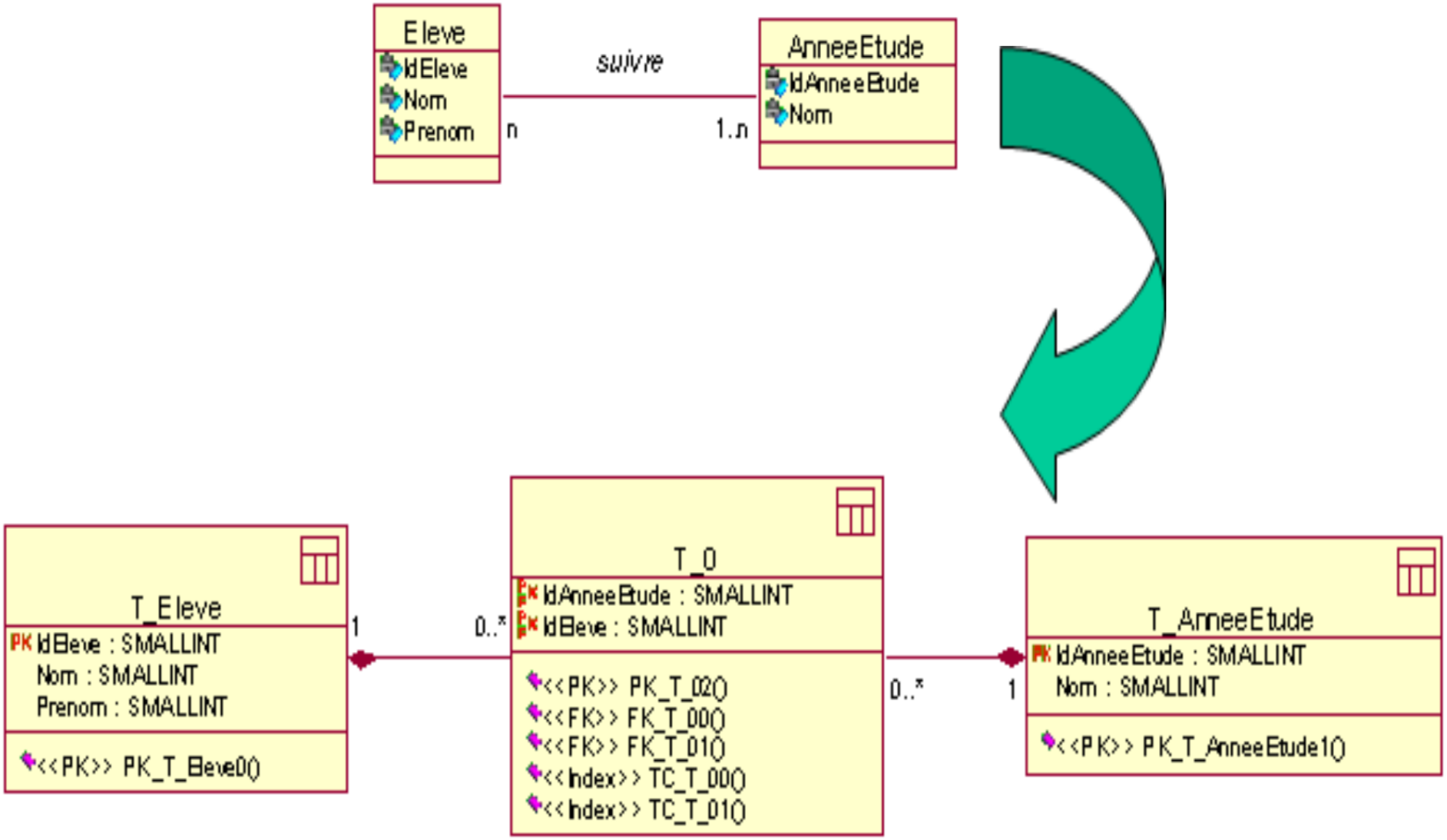
Association n-m

- On crée une **nouvelle relation** dont la clef est la concaténation des 2 clefs primaires



- Ex.: une personne peut travailler dans plusieurs entreprises
- SOCIETE(numSIRET, raisonSociale, dateCréation, ..)
- PERSONNE(numSS, nom, prénom,…)
- TRAVAIL(numSIRET, numSS)

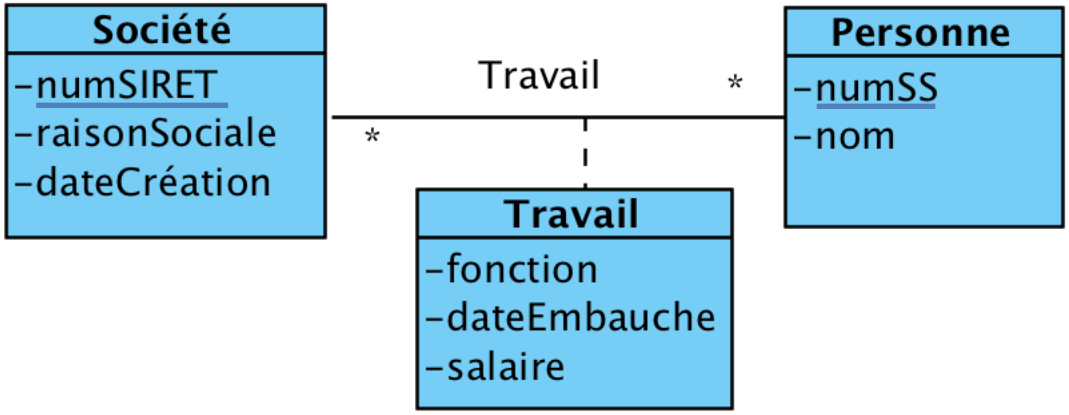
Autre exemple :



http://slideplayer.fr/slide/1301025/

Classe d'association

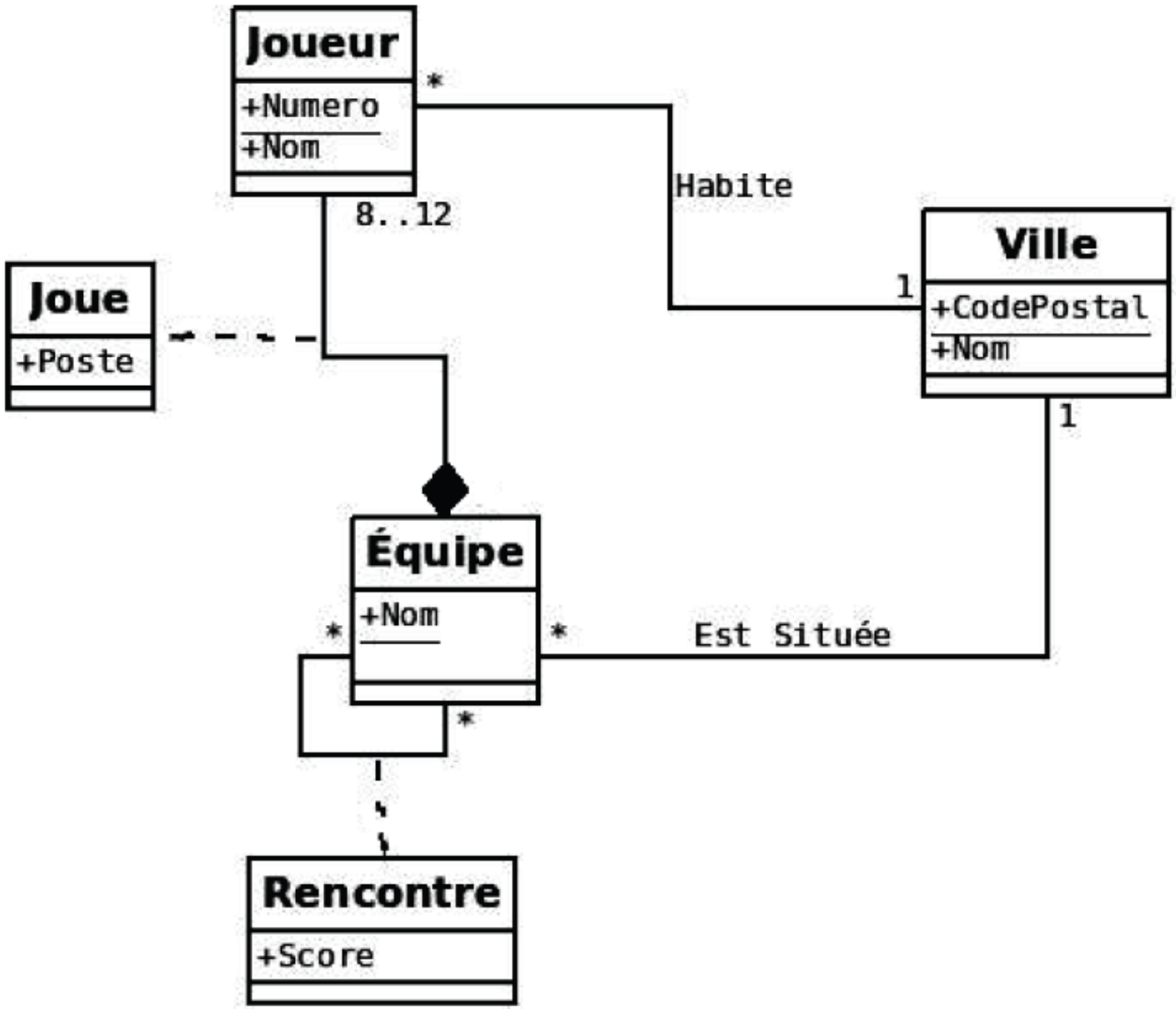
- Idem association n-m
- On ajoute les attributs de la classe d'association en plus



Ex.

TRAVAILLER(numSIRET, numSS, fonction, date Embauche, salaire)

Ex.: DCL Volley-ball



Volley : Modèle Relationnel

Joueur(#Numero, Nom, #Equipe=>Equipe, Poste, Ville=>Ville)
Joueur est une entité faible (lien de composition). Son identifiant est donc le couple NomJoueur + NomEquipe
(ici la classe d'association Joue *n'est pas légitime : cardinalité de 1 sur Equipe)*

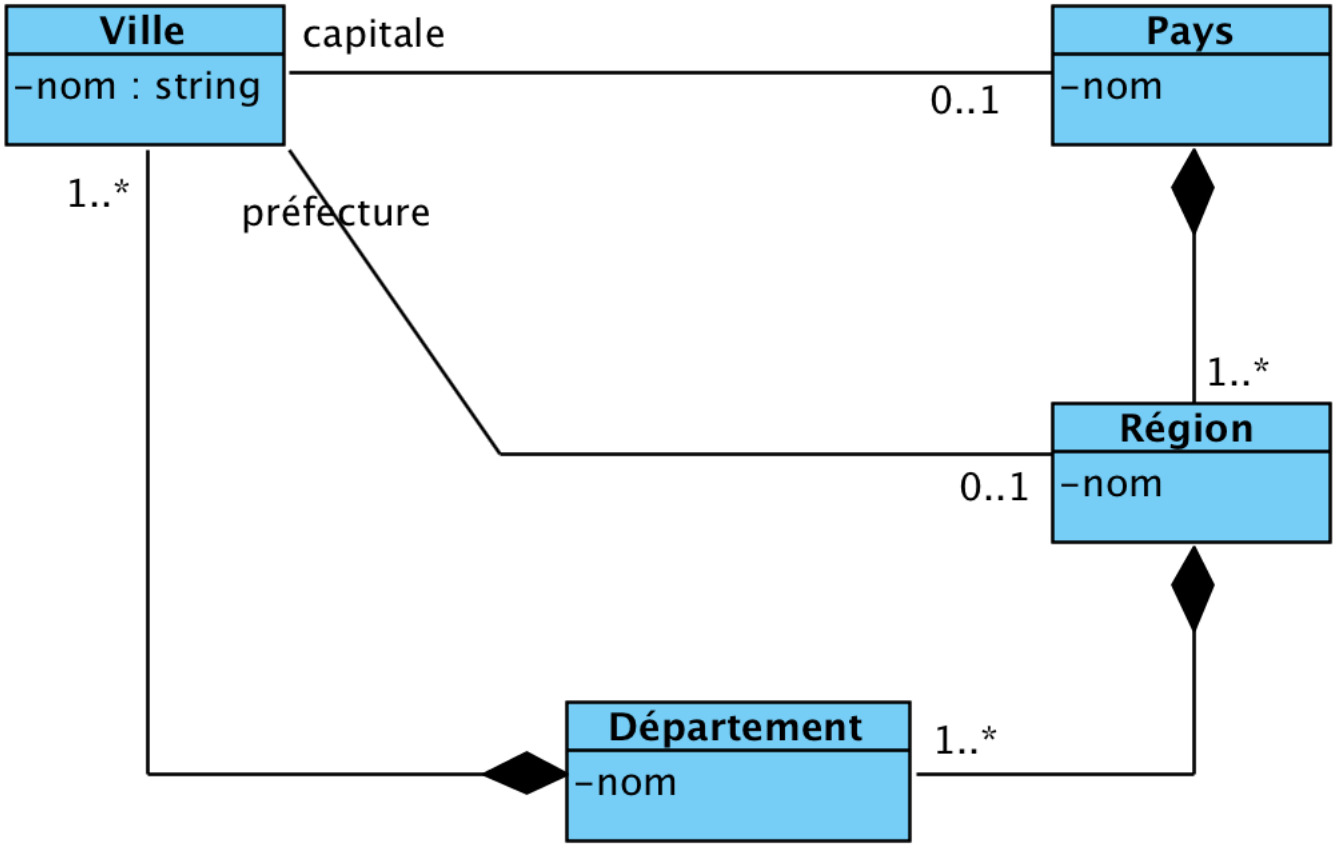
Equipe(#Nom, Ville=>Ville)

Ville(#CodePostal, Nom) // on suppose un seul nom par CP…

Rencontre(#Equipe1=>Equipe, #Equipe2=>Equipe, Score)

Exercice

Donner le MR du DCL suivant

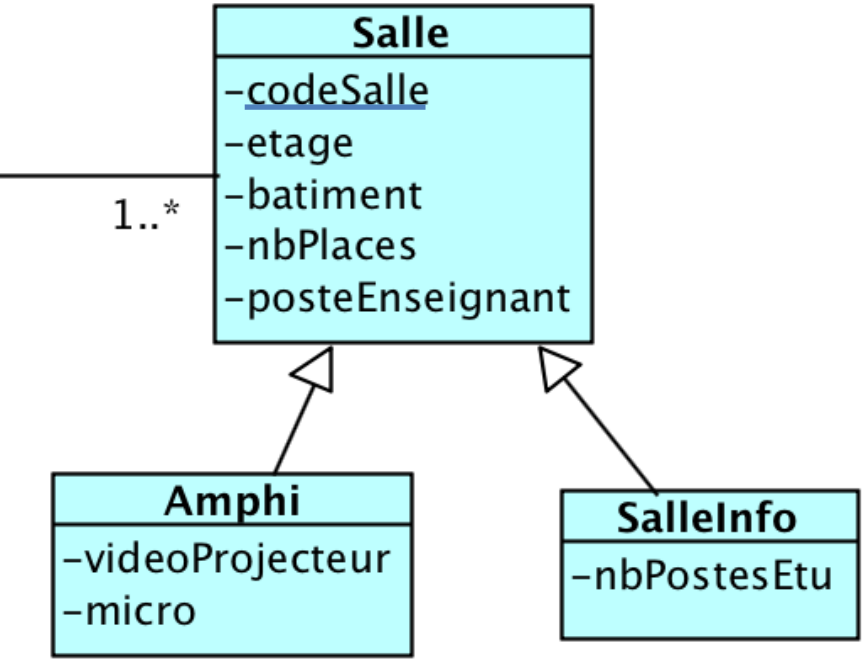


(Solution en fin de présentation)

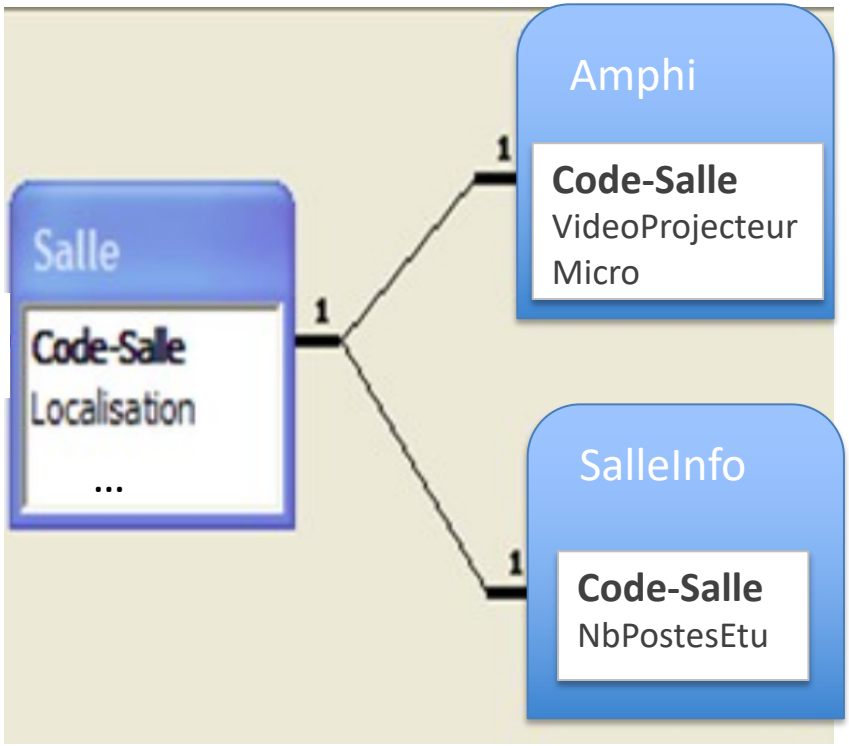
Traduction de l'héritage : méthode1

- Créer une table par sous-classe
- On utilise la clef étrangère de la classe parent pour récupérer les éléments communs
  - dans les sous-classes

Ex. DCL



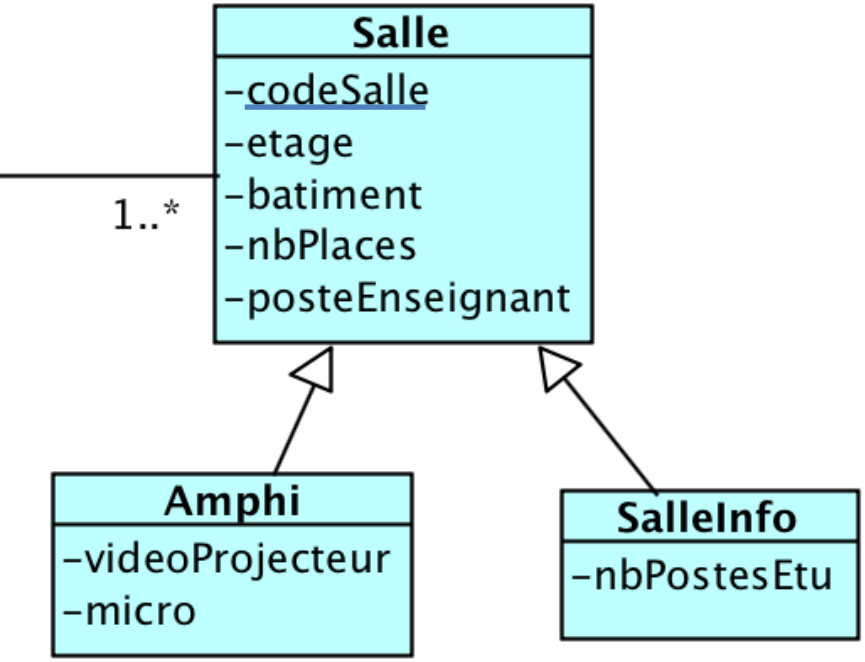
Traduction en MR



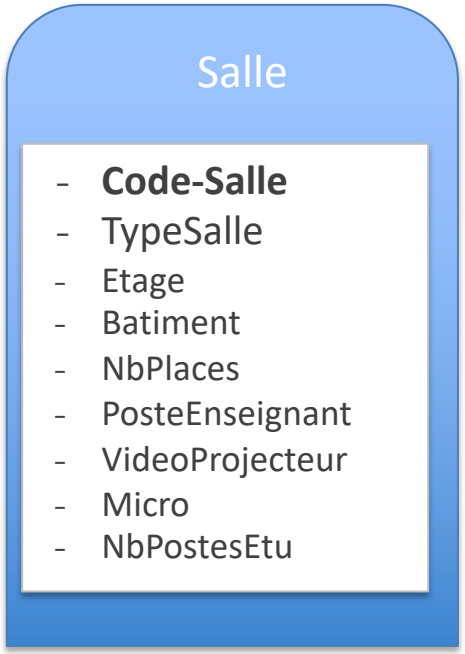
Traduction de l'héritage : méthode2

- Créer une seule table **avec tous les attributs** des sous-classes
- Ajouter un attribut permettant de distinguer les types d'objet

Ex. DCL



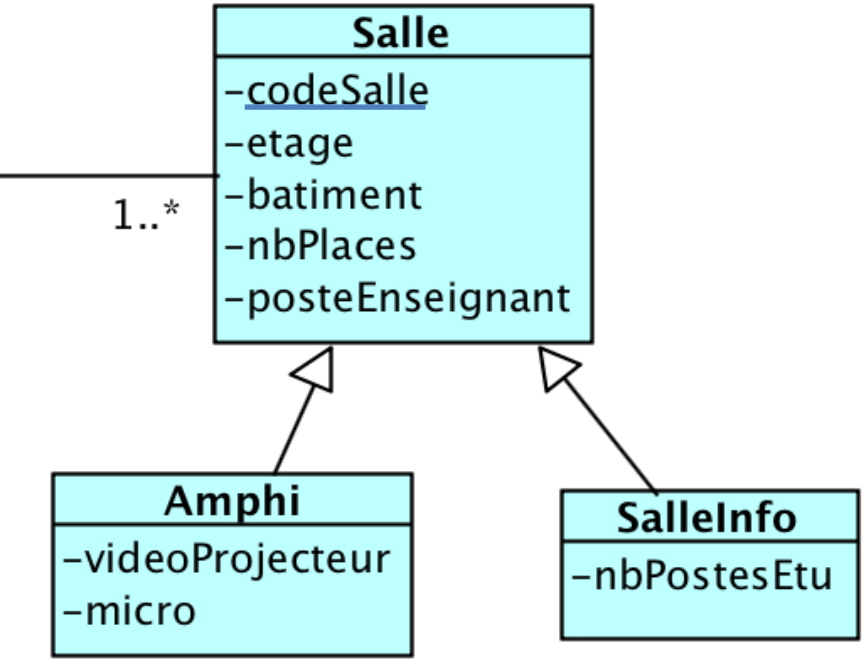
Traduction en MR



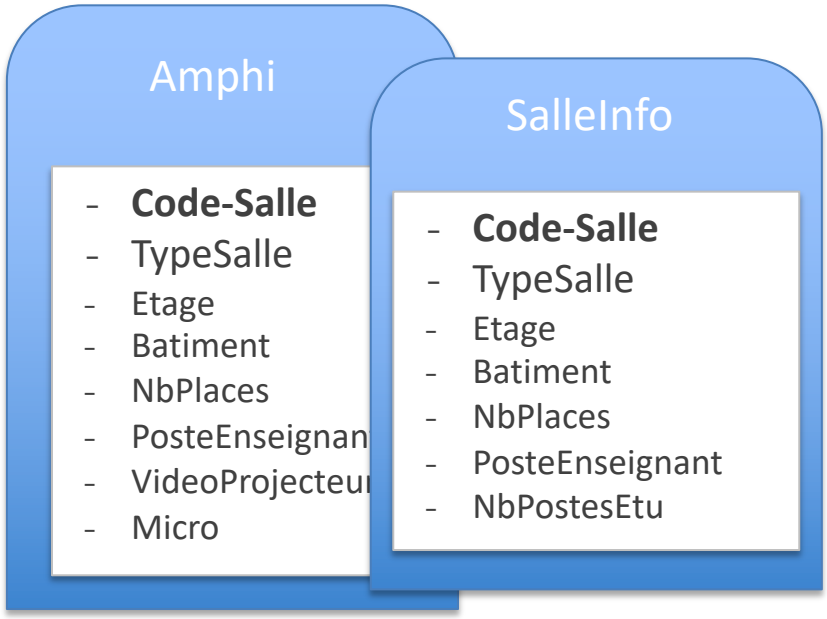
Traduction de l'héritage : méthode3

- Créer les tables correspondant aux sous-classes **avec répétitions des attributs** de la classe parent

Ex. DCL



Traduction en MR



Quel choix prendre ?

- Il dépend du volume de vos instances
- Et de la fréquence des traitements sur ces classes

- Chaque implémentation a ses avantages et inconvénients :

  https://sqlpro.developpez.com/cours/modelisation/heritage/

Clefs naturelles vs. artificielles

- Clefs artificielles :
  - Celles ajoutées pour pouvoir distinguer les enregistrements
  - Ex.: **IDLivre** (un entier) en plus de **l'ISBN**

- Les clefs artificielles sont réputées plus simples à utiliser que les clefs naturelles
  - optimisation de la base de données

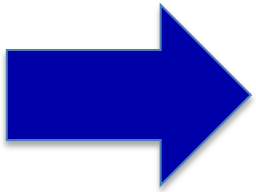
- Mais parfois, cela peut entraîner un surcroît de jointures, par ex. ici :
  - **Etu** (#id, numEtu...)
  - **UE** (#id, codeUe...)
  - **Inscriptions** (#id, ue=>UE, etu=>Etu)

Avec ce modèle, afficher la liste des étudiants (numEtu) avec leurs UEs (codeUe) nécessite une **jointure** des clefs artificielles.

Faire le choix des **clefs naturelles** aurait été plus performant ici, puisque toutes les informations se trouvaient dans la relation **Inscriptions**:

- **Etu** (#numEtu...)
- **UE** (#codeUe...)
- **Inscriptions** (#ue=>UE, #etu=>Etu)

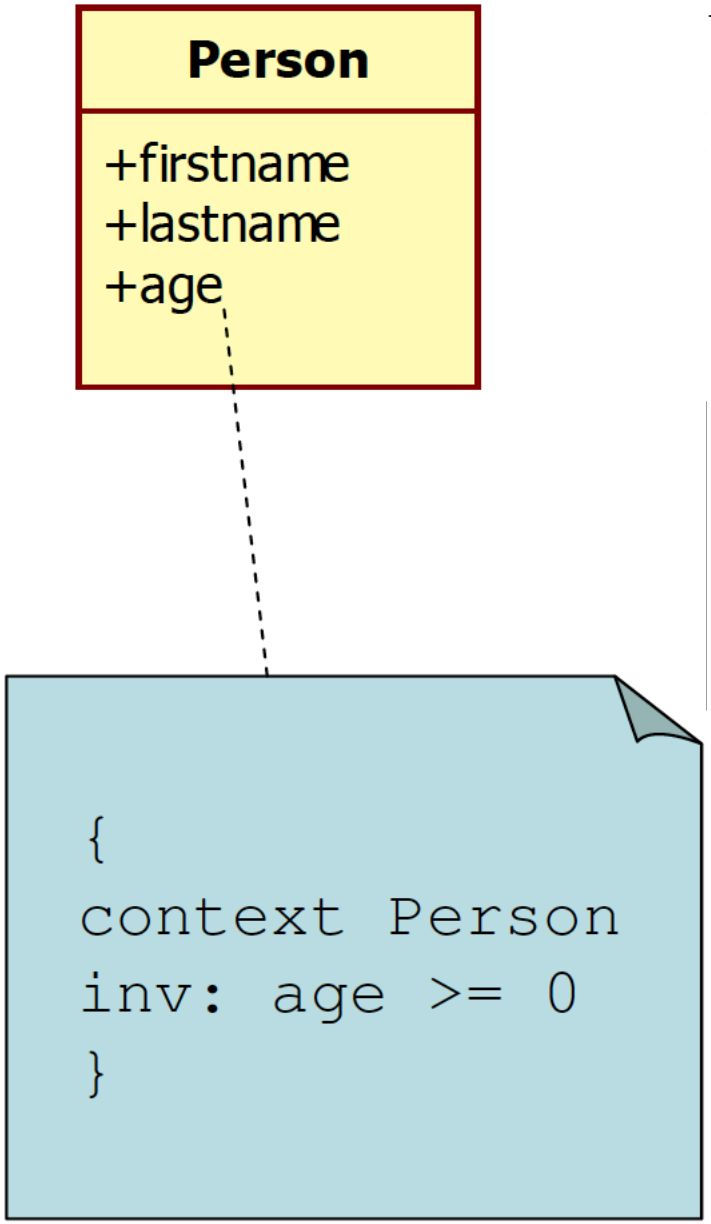
Avec un tri sur les couples (ue, etu) on a directement la liste des étudiants (numEtu) et leurs UE (ue).



DCL → JAVA

Identifiants / OID

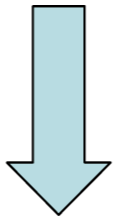
- En POO, tout objet possède un identifiant unique
  - OID (Object IDentity)
- Néanmoins, en analyse, on peut utiliser des identifiants sémantiques
  - Qui ont un sens pour le domaine du problème
  - Ex.: numéro de SS, ISBN, numéro de SIRET

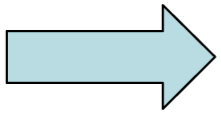



```
public class Person {
    private String firstname;
    private String lastname;
    private int age;

    public int getAge() {
        return this.age;
    }
    public void setAge(int age) {
        if(age >= 0) {
            this.age = age;
        } else {
            throw new InvalidAgeException();
        }
    }
    // ...
}
```

Traduction des associations

- Une association **navigable** revient à définir un attribut du type de l'objet relié, dans la classe d'origine

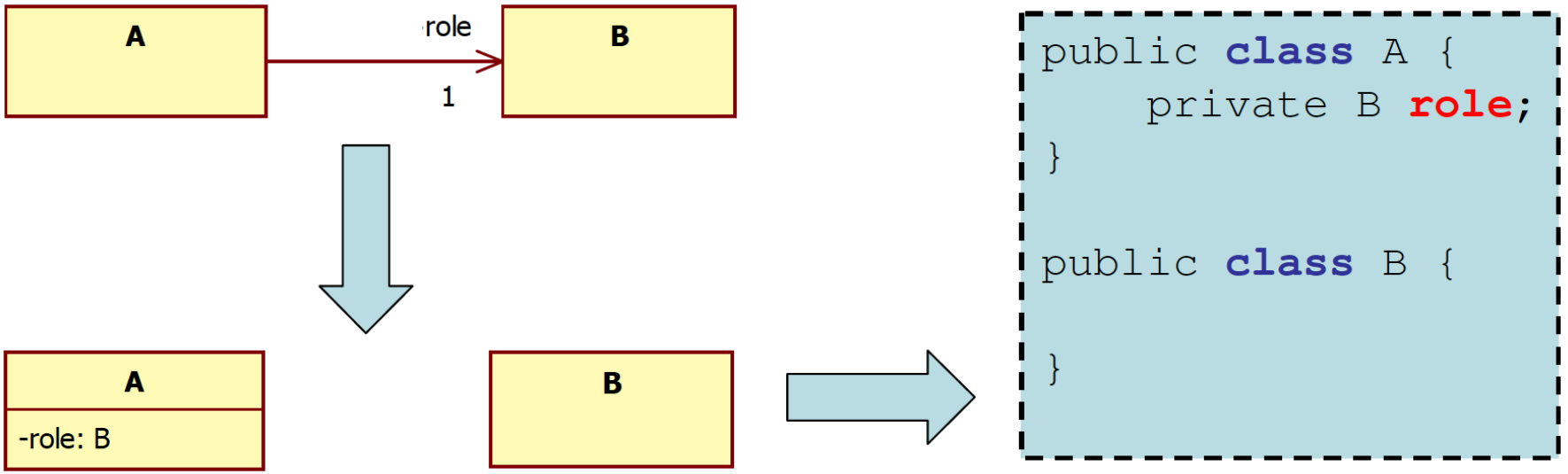




```
public class A {
    private B b;
}

public class B {

}
```

Rôle

- Utilisation du rôle de la classe s'il est mentionné



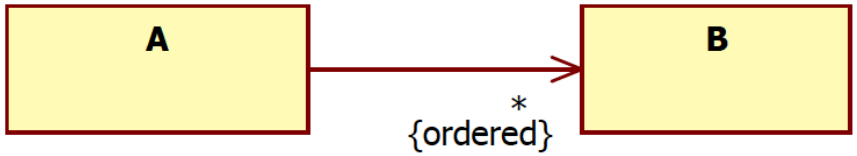
Cardinalités


```
public class A {
    private B b;
}
```

OCL `self.b : B`


```
public class A {
    private Set<B> b;
}
```

OCL `self.b : Set(B)`




```
public class A {
    private List<B> b;
}
```

OCL `self.b : OrderedSet(B)`

Get/Set

OCL `self.b : B`

Constructeur par défaut

get/set de l'objet relié

```
public class A {
    private B b;
    public A() {}
    public setB(B b) {
        this.b = b;
    }
    public B getB() {
        return b;
    }
}
```

Cas d'une composition avec plusieurs objets


```
public class A {
    private Set<B> b;

    public A() {
        b = new HashSet<B>();
     // b = new … implements Set<>
         }

    public A(Set<B> b) {
        this.b = b;
    }
```

OCL `self.b : Set(B)`

b est créé dans les constructeurs (cycles de vie liés)

Constructeur par copie

Suite… get / set


```
public class A {
    private Set<B> b;
```

OCL `self.b : Set(B)`

```
public Set<B> getB() {
    return this.b;
}
```

get b

Ajout des objets b, un ou plusieurs

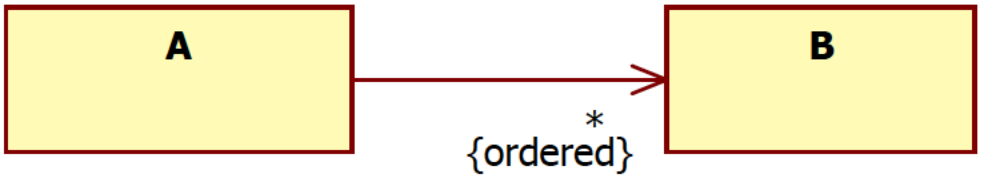
```
    public boolean add(B b) {
        return this.b.add(b);
    }
    public boolean addAll(B... b) {
        return this.b.addAll(Arrays.asList(b));
    }
    public boolean addAll(Collection<B> b) {
        return this.b.addAll(b);
    }
}
```

Et dans le main()…

OCL `self.b : Set(B)`

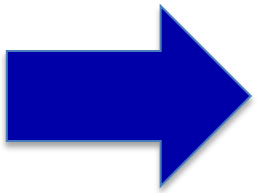
```
main() {
   B b1 = new B();
   ….
   A a = new A();
   a.add(b1);
   a.add(new B());
}
```

Association *{ordered}*



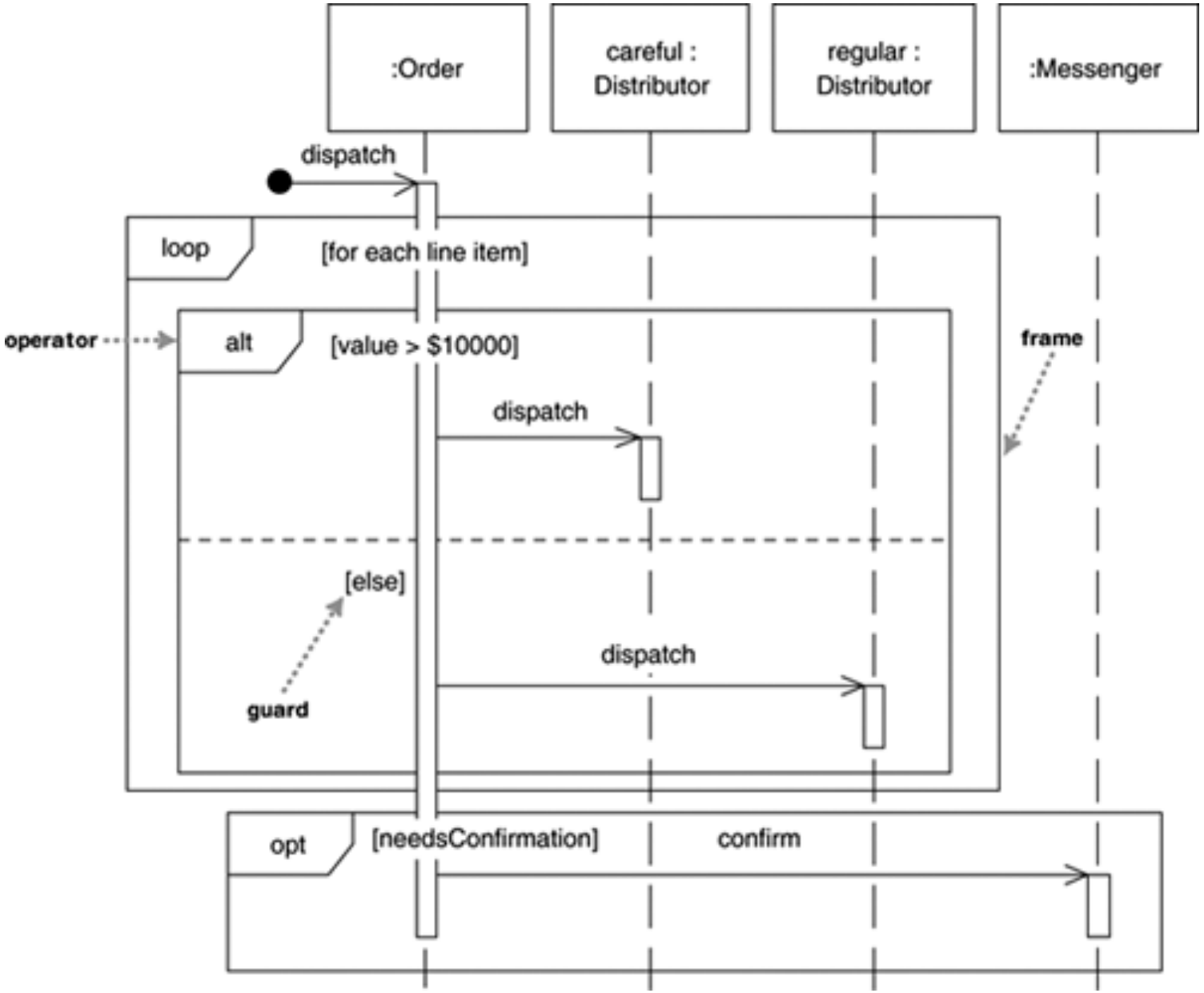
OCL `self.b : OrderedSet(B)`

```
public class A {
    private List<B> b;

    public A() {
        b = new ArrayList<B>();
// b = new LinkedList<B>();
```



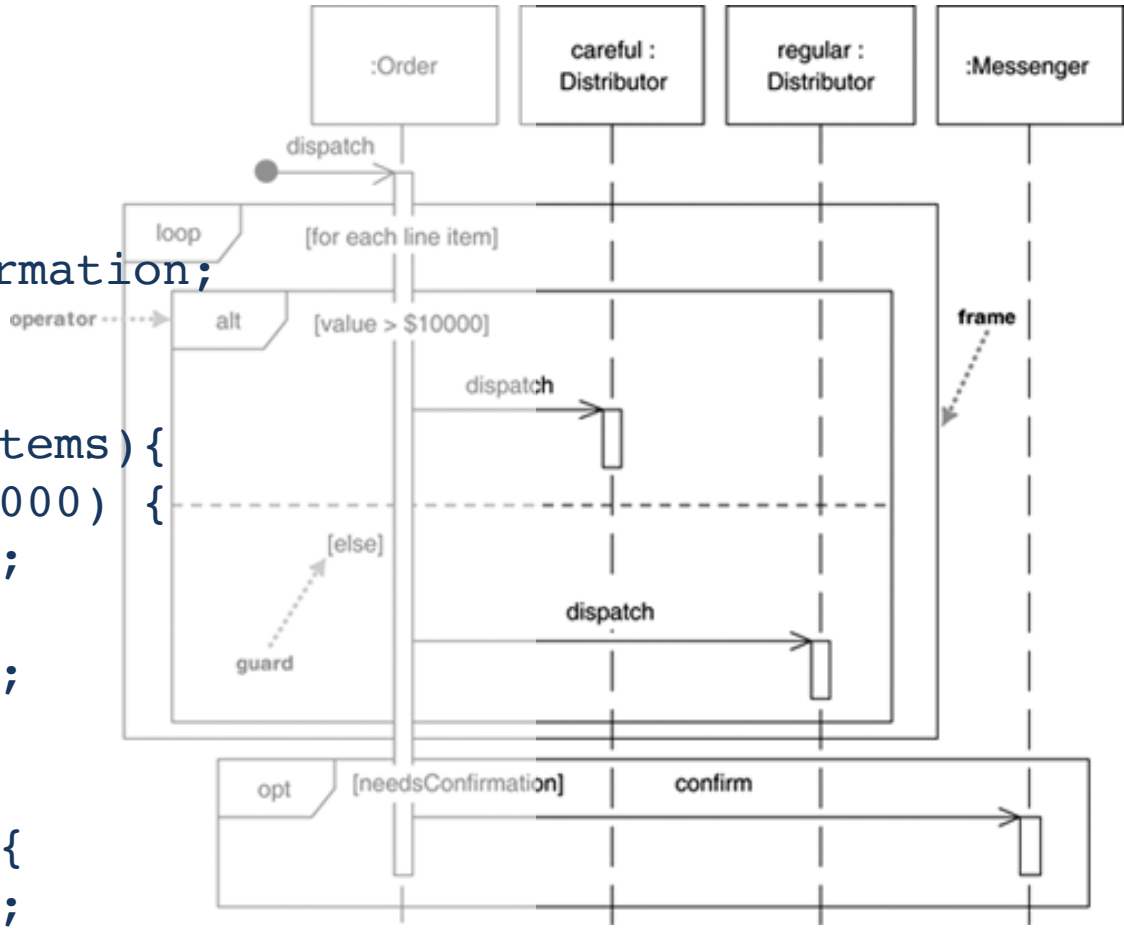
DSQ → JAVA

Traduction d'un DSQ



Ce qui donne en Java

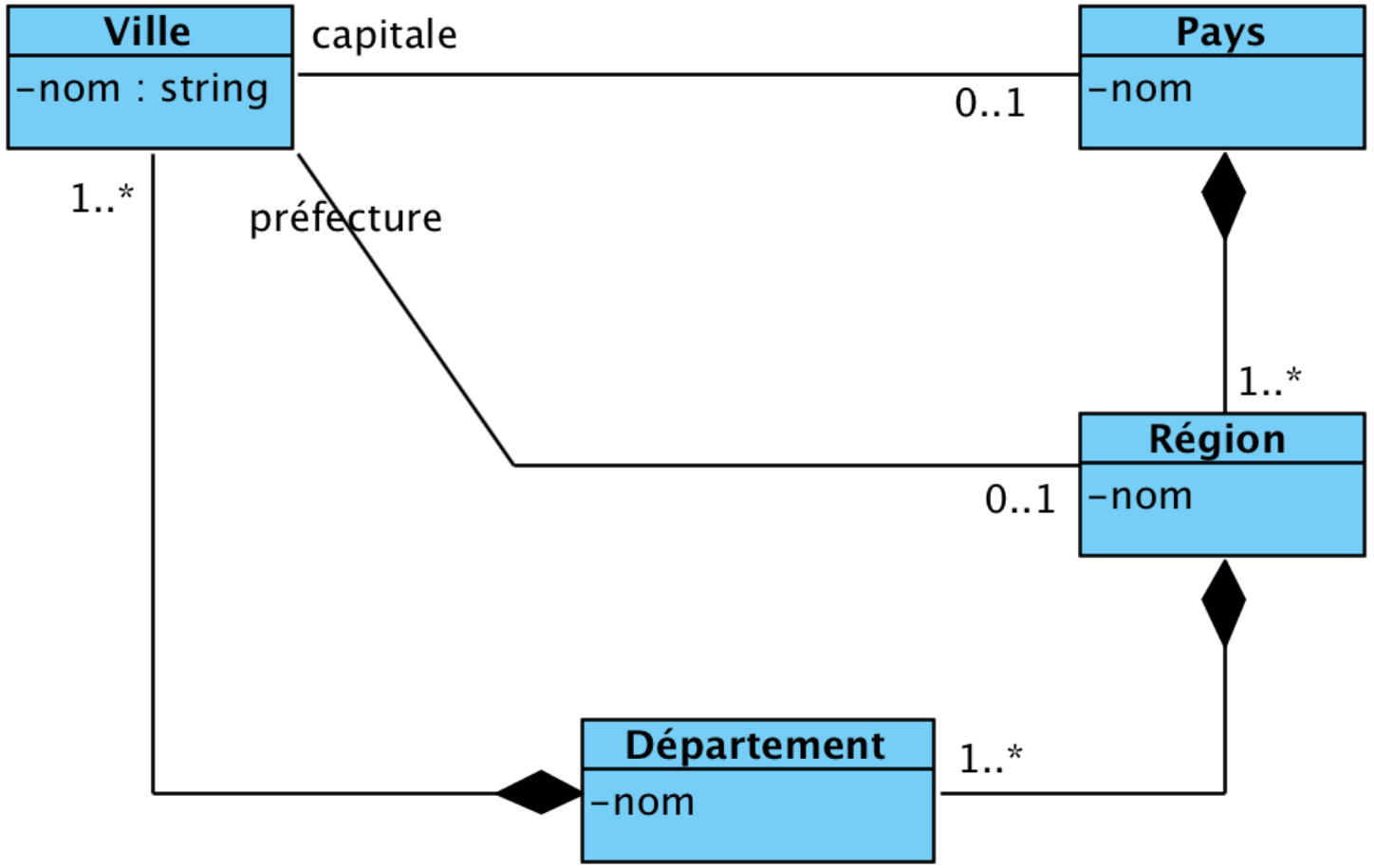
```
public class Order {
  List<LineItem> lineItems;
  Distributor    careful;
  Distributor    regular;
  Messenger      messenger;
  boolean        needsConfirmation;

  public void dispatch() {
    for(LineItem li : lineItems){
      if(product.value > 10000) {
         careful.dispatch();
      } else {
         regular.dispatch();
      }
    }
    if(needsConfirmation) {
       messenger.confirm();
    }
  }
}
```



[Martin Fowler, 2003]

Corrigé de l'Exercice

Donner le MR du DCL suivant



(Solution en fin de présentation)

Corrigé MR Région Capitale

Ici toutes les entités *sauf Pays* sont faibles (lien de composition). Elles héritent donc de la clé primaire de leur entité identifiante :

Pays(#Nom, Nom, CapitaleVille=>Ville, CapitaleDepartement=>Ville, CapitaleRegion=>Ville, CapitalePays=>Ville)

Region(#Nom, , #Pays=>Pays, PrefectureVille=>Ville, PrefectureDeparement=>Ville, PrefectureRegion=>Ville, PrefecturePays=>Ville)

Departement(#Nom, #Region=>Region, #Pays=>Region)

Ville(#Nom, #Departement=>Departement, #Region=>Departement, #Pays=>Departement)