Architecture Multi-tiers

Relationnel Object Mapping/ Mapping Objet Relationnel

Plan

Mapping Objet Relationnel

- Introduction et problématique
- Architecture multi-niveaux
- Correspondance Objet/Relationnel

Introduction et problématique

- La tendance actuelle de développement se fait dans une **approche objet** (Langages et méthodes) et utilisant une **base de données relationnelle**.
- Les développeurs confrontés à deux problèmes :
  - Le problème de choix de la plateforme de développement, et celui du serveur de bases de données
  - Cohabitation entre les mondes objets et relationnels

Introduction et problématique

Pour le premier problème (plateforme de développement et serveur de bases de données)

- J2EE/Oracle
- .Net (Asp.Net,C#,VB.Net,…)/SQL Server
- ASP/Access
- PHP/MySQL
- etc.

Introduction et problématique

Pour le deuxième problème :

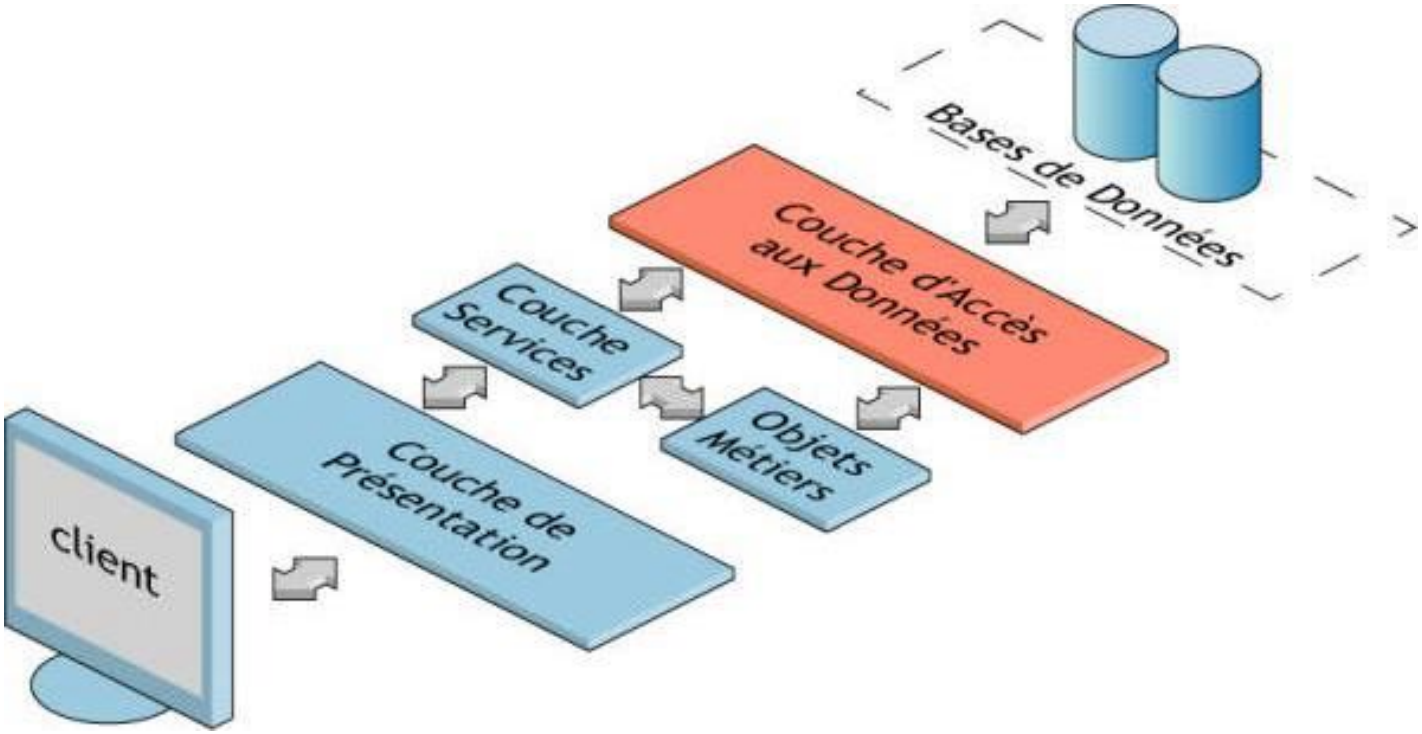
- La cohabitation entre les mondes objets et relationnels est résolue grâce au concept de **Mapping objet-relationnel**
- **Le Mapping** est le nom donné aux techniques de transformation des modèles objets en modèles relationnels.
- Parmi ces techniques, on trouve :
  - Règles (Manuelle)
  - Outils (Hibernate, JGrinder, Persistent Database, …)
  - …

Architecture multi-niveaux

- Aujourd'hui tout le monde opte pour une séparation en différentes couches des applications, et parlent alors d'applications multi niveaux (n-tier applications).
- plusieurs modèles ont vu le jour
  - Seeheim
  - PAC
  - Hudson
  - Arch
  - MVC
  - …

Architecture multi-niveaux

En général, l'architecture présente ces couches :



Architecture multi-niveaux

Avantages

- Couches indépendantes ➔ modification d'une couche n'affecte pas les autres
- Le développement pourra se faire en parallèle
- La maintenance de l'application n'en sera que plus aisée.
- Rendre l'accès aux données complètement indépendant du SGBD utilisé. Il devient donc très simple de changer de SGBD au cours du développement de l'application

Architecture multi-niveaux

- Couche de présentation :
  - constitue la partie qui est en contact direct les médias de sortie
  - correspond à l'interface avec laquelle l'utilisateur interagit
  - responsable de la représentation externe sous forme textuelle et graphique des objets internes et abstraits de l'application
  - **correspond à la couche Vue du modèle MVC**

Architecture multi-niveaux

- couche métier :
  - stocke les données de l'application
  - contient les objets métiers de l'application
  - avant son implémentation, cette couche fait l'objet d'une conception approfondie en élaborant les différents diagrammes UML, notamment le diagramme de classes
  - **correspond à la couche Modèle du modèle MVC**

Architecture multi-niveaux

- Couche service
  - Traite les entrées de l'utilisateur
  - Il informe la couche métier des interactions faites par l'utilisateur
  - La couche métier modifie alors son état et informe la couche présentation du nouvel aspect qu'elle doit prendre suite à cette modification
  - **Correspond à la couche Contrôleur du modèle MVC**

Architecture multi-niveaux

- La couche d'accès aux données doit prendre en charge toutes les interactions entre l'application et la base de données.
  - la création
  - la lecture
  - la modification
  - la suppression

  des enregistrements dans chacune des tables de la base de données.
- Généralement, on créée pour chaque classe du modèle métier une classe correspondante dans la couche d'accès aux données

Correspondance Objet Relationnel

- Cette partie traite le passage de la conception Objet (faite par UML) vers :
  - le codage en Objet : en C++ ou en Java
  - Modèle Relationnel / SQL
- Ce passage concerne
  - Classes, instances de classes
  - Attributs et opérations de classes ou d'instances
  - Relations entres classes : associations, agrégation, composition et généralisation spécialisation

Classes, instances, attributs et opérations en Objet

Objet (UML)	C++	Java
Classe	Classe	Classe
Instance	Instance	Instance
Attribut	Membre	Attribut
Opération	Fonction	Méthode

Attributs et opérations de classe ou d'instance en Objet

	Objet (UML)	**C++**	**Java**
Attribut et opération de classe	Souligné	static	static
Attribut et opération d'instance	-	-	-

Classe en Relationnel

- Dans le cas général une classe est traduite par une table
- Chaque objet est conservé dans une ligne de la table
- Un champ jouant le rôle de clé primaire est ajouté même s'il n'existait pas dans la classe

Traduction d'une classe

- **En Java**

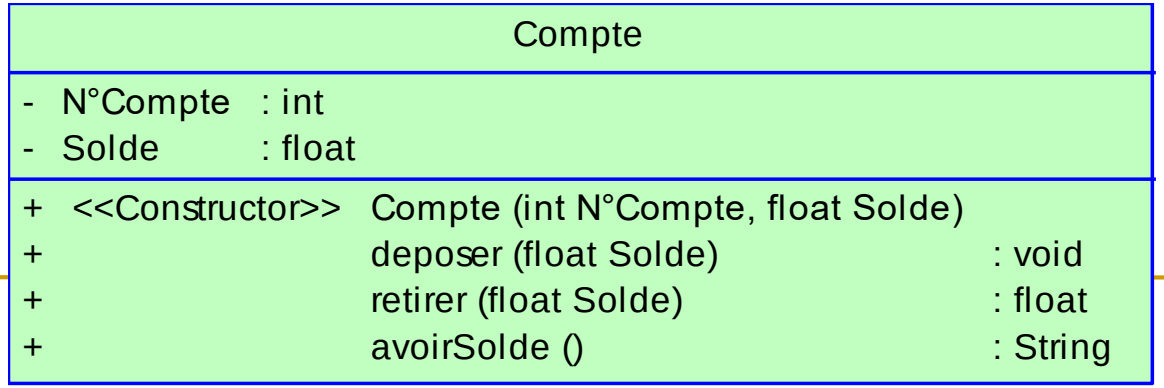
```
class Compte{
    private int NCompte;
    private float Solde;
    public Compte(int NC, float S){
    …
    }
    public void deposer(float Solde){
    …
    }
    public String avoirSolde(){
    …
    }
}
```

- **En Relationnel**

Compte(**NCompte**, Solde)

- **En SQL**

```
Create table Compte(
    NCompte smallint,
    Solde decimal,
    Primary key PK_Compte
      (NCompte)
)
```



Généralisation/spécialisation en Objet

- La relation de généralisation/spécialisation entre classes est la plus immédiate
- Elle est traduite par une relation d'héritage dans l'implémentation
- Chaque langage utilise sa propre syntaxe :
  - Class A extends B{…} en Java
  - Class A : public B{ …} en C++

Généralisation/spécialisation en Objet

En Java :

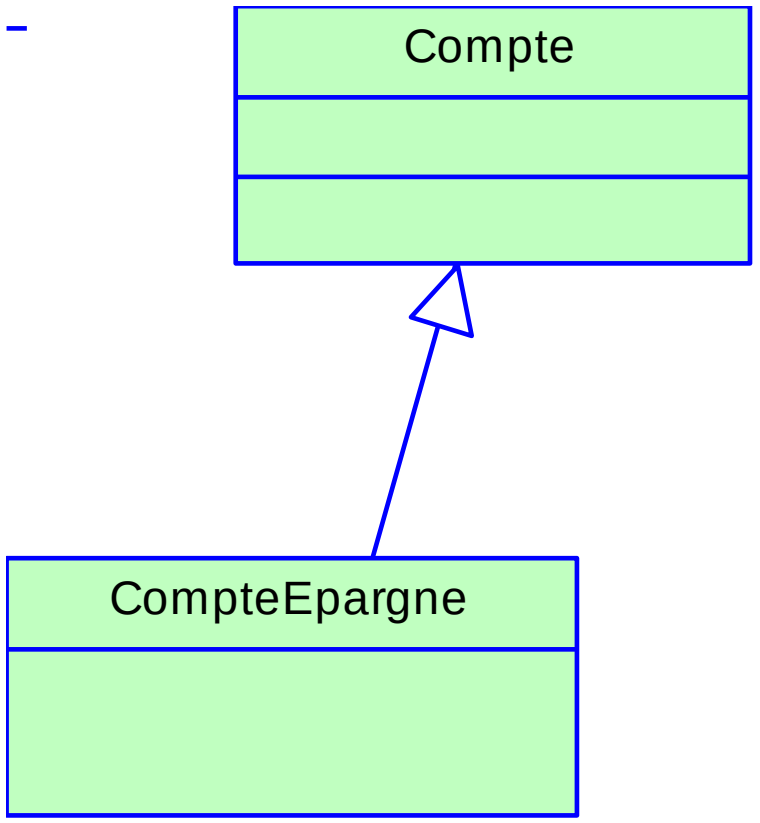
class Compte{…}

class CompteEpargne
    extends Compte{…}

En C++ :

class Compte{…};

class CompteEpargne :
    public Compte{…};



Généralisation/spécialisation en Relationnel

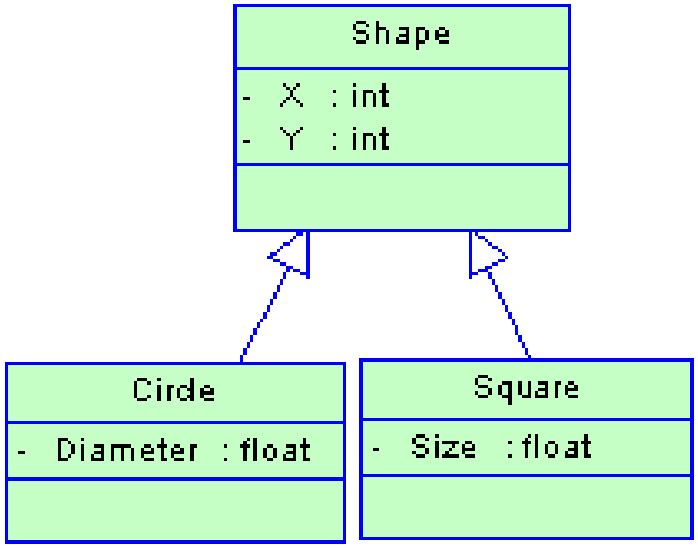
Plusieurs méthodes de traduction en Relationnel :

- Représenter toutes les classes d'une arborescence d'héritage par une seule table relationnelle
- Représenter chaque classe instanciable par une table

Généralisation/spécialisation en Relationnel

- La solution la plus simple est de modéliser toute une hiérarchie de classes dans une même table
- Chaque classe ajoutant ses propres attributs comme de nouveaux champs.
- Il nous suffit alors d'ajouter un champ contenant le type de l'instance pour pouvoir charger les champs correspondants.

Généralisation/spécialisation en Relationnel



	ID	Type	X	Y	Diameter	Size
	1	Circle	10	20	10,5	<NULL>
	2	Circle	10	15	5,1	<NULL>
	3	Square	5	5	<NULL>	13,2
▶						

Associations en Objet

- Plusieurs façons pour traduire les associations UML en Objet
- Une association UML se traduira par un ou plusieurs attributs : pointeurs ou références, vers un ou plusieurs objets
- On distingue trois cas
  - Association un à un
  - Association un à plusieurs
  - Association plusieurs à plusieurs

Associations en Objet (Cas d'une association un-à-un)

- Les deux rôles de l'association seront traduits par :
  - des pointeurs en C++
  - des références en Java
- Les pointeurs ou références peuvent être ajoutés soit dans l'une des classes soit dans les deux classes (suivant la navigabilité)

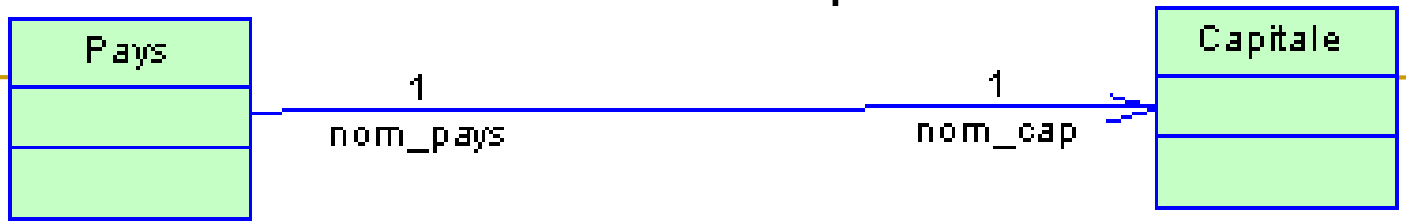
Associations en Objet
(Cas d'une association un-à-un)

En C++ :

```
class Pays{
…
Capitale *nom_cap;
…
};
class Capitale{
…
};
```

En Java :

```
class Pays{
…
Capitale nom_cap;
…
}
class Capitale{
…
}
```



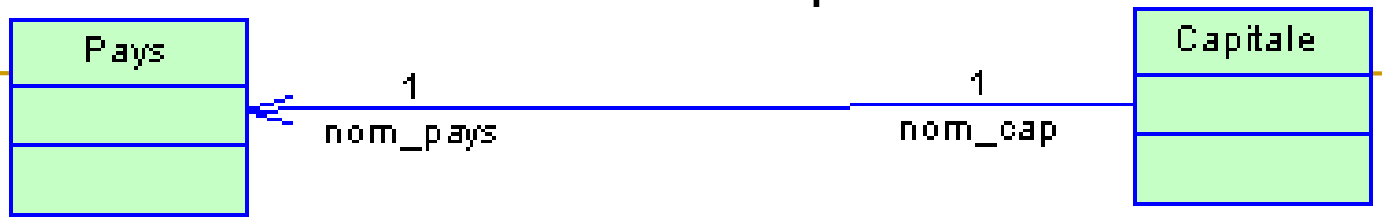
Traduction des associations en Objet (Cas d'une association un-à-un)

En C++ :

```
class Pays{
…
};
class Capitale{
…
Pays *nom_pays;
…
};
```

En Java :

```
class Pays{
…
}
class Capitale{
…
Pays nom_pays;
…
}
```



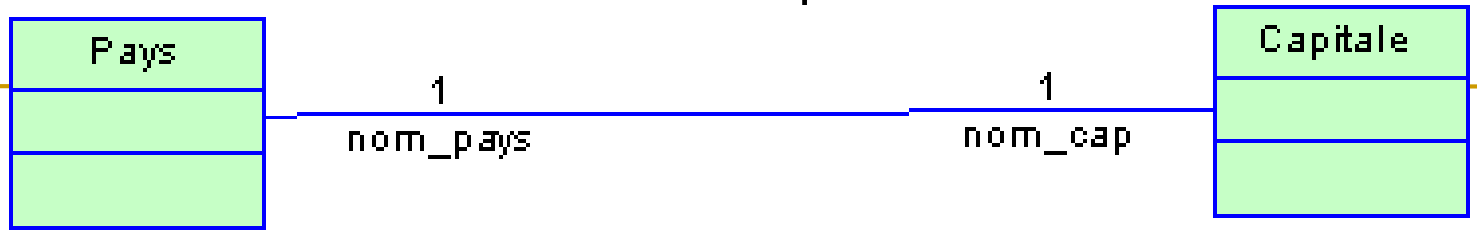
Traduction des associations en Objet (Cas d'une association un-à-un)

En C++ :

```
class Pays{
…
Capitale *nom_cap;
…
};
class Capitale{
…
Pays *nom_pays;
…
};
```

En Java :

```
class Pays{
…
Capitale nom_cap;
…
}
class Capitale{
…
Pays nom_pays;
…
}
```



Traduction des associations en Objet (Cas d'une association un-à-plusieurs)

Dans le code objet, une association peut être représentée par :

- une variable d'instance représentant "l'autre" objet avec lequel se fait l'association
- une variable d'instance de type collection représentant tous les autres objets avec lesquels se fait l'association

Associations en Objet (Cas d'une association un-à-plusiuers)

Dans le cas où nous avons besoin d'un attribut pouvant contenir un nombre quelconque d'éléments : Un tableau n'est pas adapté car il possède une taille qui limite le nombre d'éléments

- En C++, on utilise un pointeur sur un pointeur
- En Java, on préfère utiliser une collection (Vector, Iterator, …)

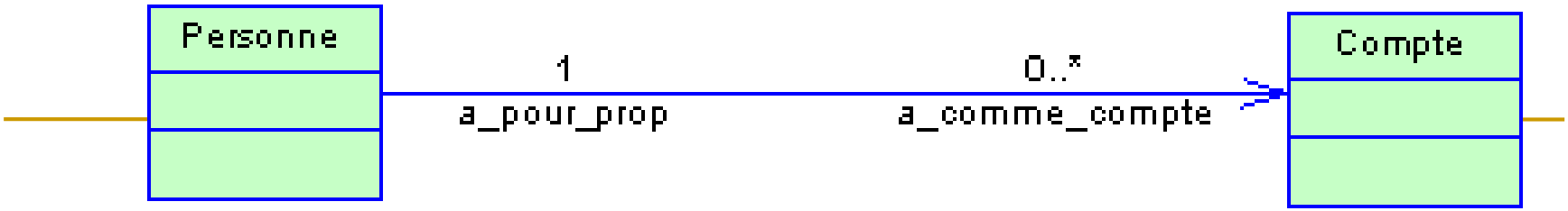
Associations en Objet
(Cas d'une association un-à-plusiuers)

En C++ :

```
Class Personne{
…
Compte **a_comme_compte;
…
}
Classe Compte{
…
}
```

En Java :

```
Class Personne{
…
Vector a_comme_compte;
…
}
Class Compte{
…
}
```



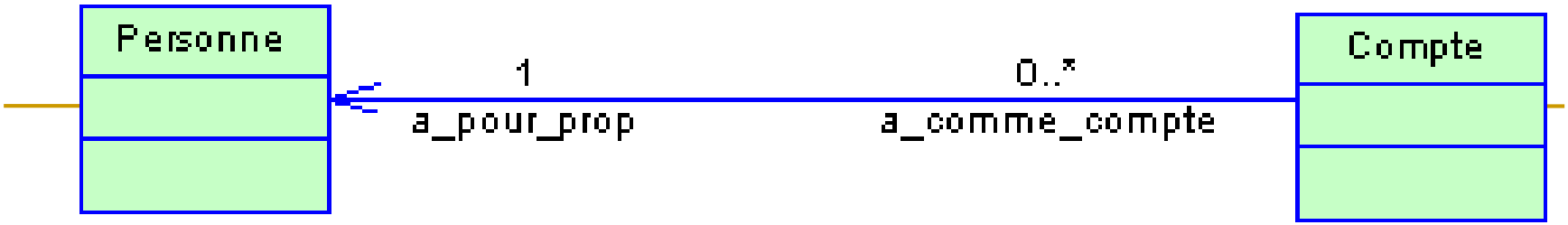
Associations en Objet (Cas d'une association un-à-plusiuers)

En C++ :

```
Class Personne{
…
}
Classe Compte{
…
Personne *a_pour_prop;
…
}
```

En Java :

```
Class Personne{
…
}
Class Compte{
…
Personne a_pour_prop;
…
}
```



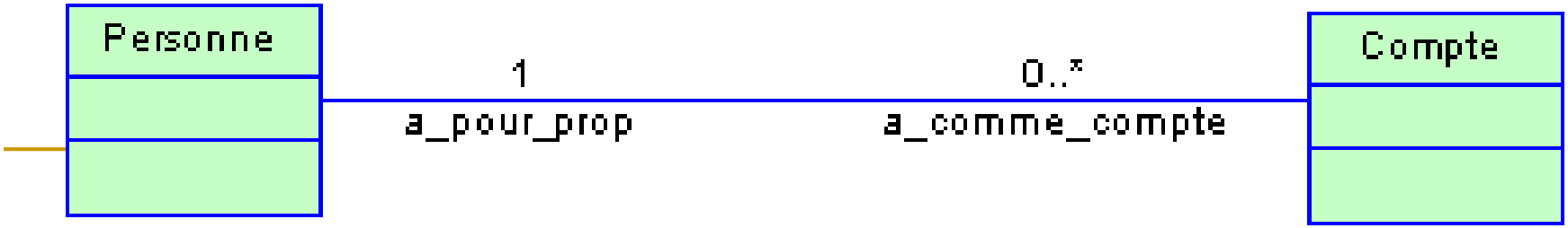
Associations en Objet (Cas d'une association un-à-plusiuers)

En C++ :

```
Class Personne{
…
Compte **a_comme_compte;
…
}
Classe Compte{
…
Personne *a_pour_prop;
…
}
```

En Java :

```
Class Personne{
…
Vector a_comme_compte;
…
}
Class Compte{
…
Personne a_pour_prop;
…
}
```

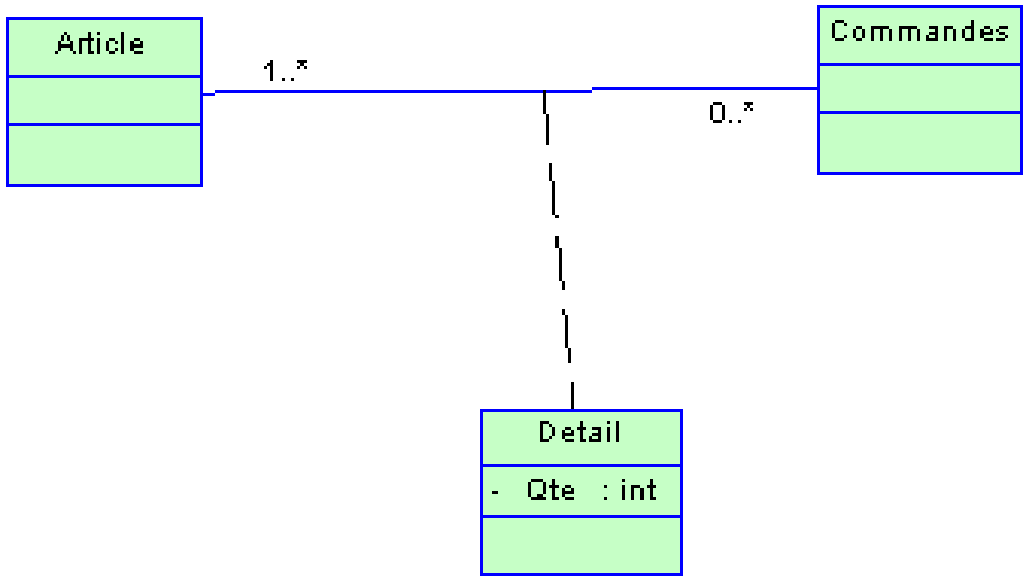


Associations en Objet (Association plusieurs-à-plusieurs)

- On distingue ici deux cas :
  - Association porteuse d'informations (classe association) : la classe association est implémentée par une classe contenant en plus de ses informations, les références ou les pointeurs des classes associées
  - Association non porteuse d'informations : deux solutions :
    - les références ou pointeurs se migrent vers l'une ou les deux classes associées
    - création d'une classe contenant les références ou les pointeurs des classes associées

Associations en Objet (Association plusieurs-à-plusieurs)

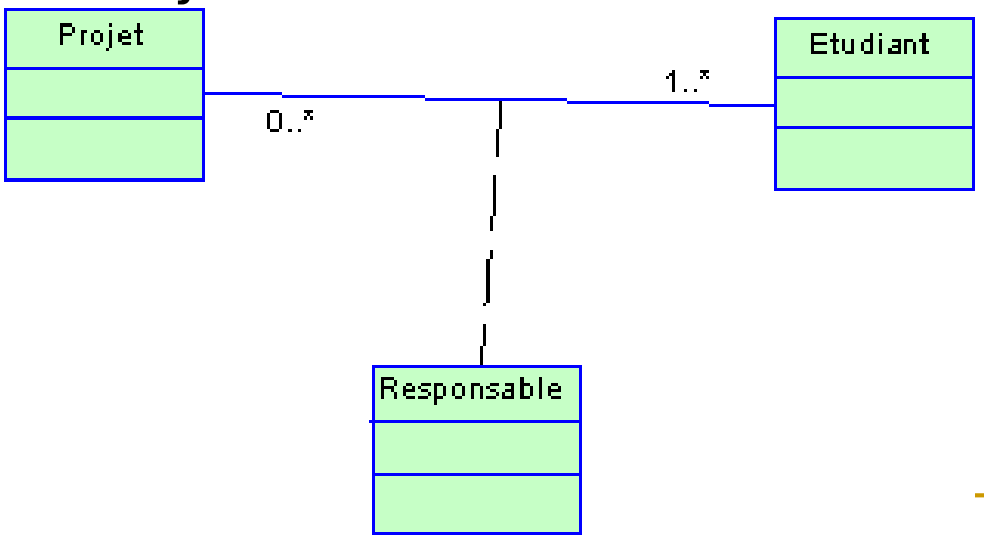
```
class Detail{
    private Article Contenu;
    private Commandes Conteneur;
    private int Qte;
    …
}
```



Associations en Objet (Association plusieurs-à-plusieurs)

```
class Responsable{
    private Projet NP;
    private Etudiant Code;
    ….
    }
```

```
Class Projet{
    …
    private Etudiant Code;
    …
    }
class Etudiant{
    …
    private Projet NP;
    …
    }
```



Agrégation et composition en Objet

En C++ :

- L'agrégation est implémentée comme l'association
- La composition sera traduite par un attribut de composant dans la classe composée

En Java :

- L'agrégation, composition et association sont traduites de la même manière

Associations en Relationnel (Association un-à-un)

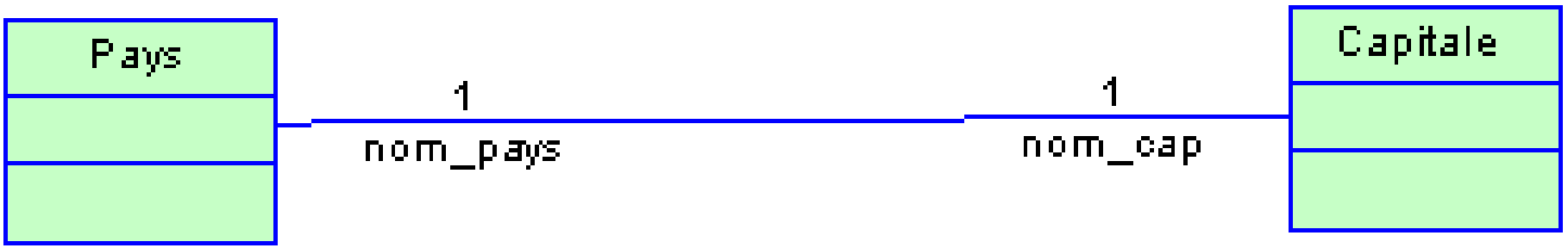
Deux solutions sont possibles :

- une clé étrangère dans chacune des tables associées
- la fusion des deux tables dans une seule

Associations en Relationnel (Association un-à-un)

- 1ère Solution
  - Pays(**IdPays**, NomP,#IdCapitale)
  - Capitales(**IdCapitale**, NomC, #IdPays)

- 2ième Solution
  - Pays(**IdPays**, IdCapitale, NomP, NomC)



Associations en Relationnel (Association un-à-un)

- 1ère Solution

  create table Pays(IdPays integer primary key,

  …

  IdCapitale integer foreign key references capitales (IdCapitale))

et

  create table Capitales(IdCapitale integer primary key,

  …,

  IdPays integer foreign key refernces pays(IdPays))

- 2ième Solution

  Pays(IdPays integer primary key, IdCapitale integer,

  NomP varchar(20),

  NomC varchar(20))

Associations en Relationnel (Association un-à-plusieurs)

Une seule solution est possible :

- migration de la clé du côté de 1 vers la table du côté de plusieurs
- La clé migrée jouera le rôle de clé étrangère

Associations en Relationnel (Association un-à-plusieurs)

- En Relationnel
  - Dept(**IdDept**, Nomdept)
  - Emp(**IdEmp**, NomEmp, #IdDept)
- En SQL
  - Create table dept(…)
  - Create table emp(IdEmp integer primary key,
    NomEmp varchar(20),
    IdDept integer foreign key references Dept(IdDept)
)

Associations en Relationnel (Association plusieurs-à-plusieurs)

- L'association est traduite par une table dont la clé primaire est la concaténation des clés primaires des tables associées
- La table résultante aura :
  - Une seule clé primaire
  - Deux clés étrangères

Traduction des associations en Relationnel (Association plusieurs-à-plusieurs)

En Relationnel

- Articles(**Ref**, Des, PU)
- Commandes(**NBC**, DateC, Client)
- Détails(#**NBC, #Ref**, Qté)

Traduction des associations en Relationnel (Association plusieurs-à-plusieurs)

En SQL

1. create table Article(Ref integer primary key, …)
2. create table Cde(NBC integer primary key, …)
3. create table Detail(NBC integer, Ref integer,…,
   constraint PK primary key(NBC, Ref),
   constraint FK1 foreign key(NBC) references cdes(NBC),
   Constraint FK1 foreign key(NBC) references cdes(NBC))