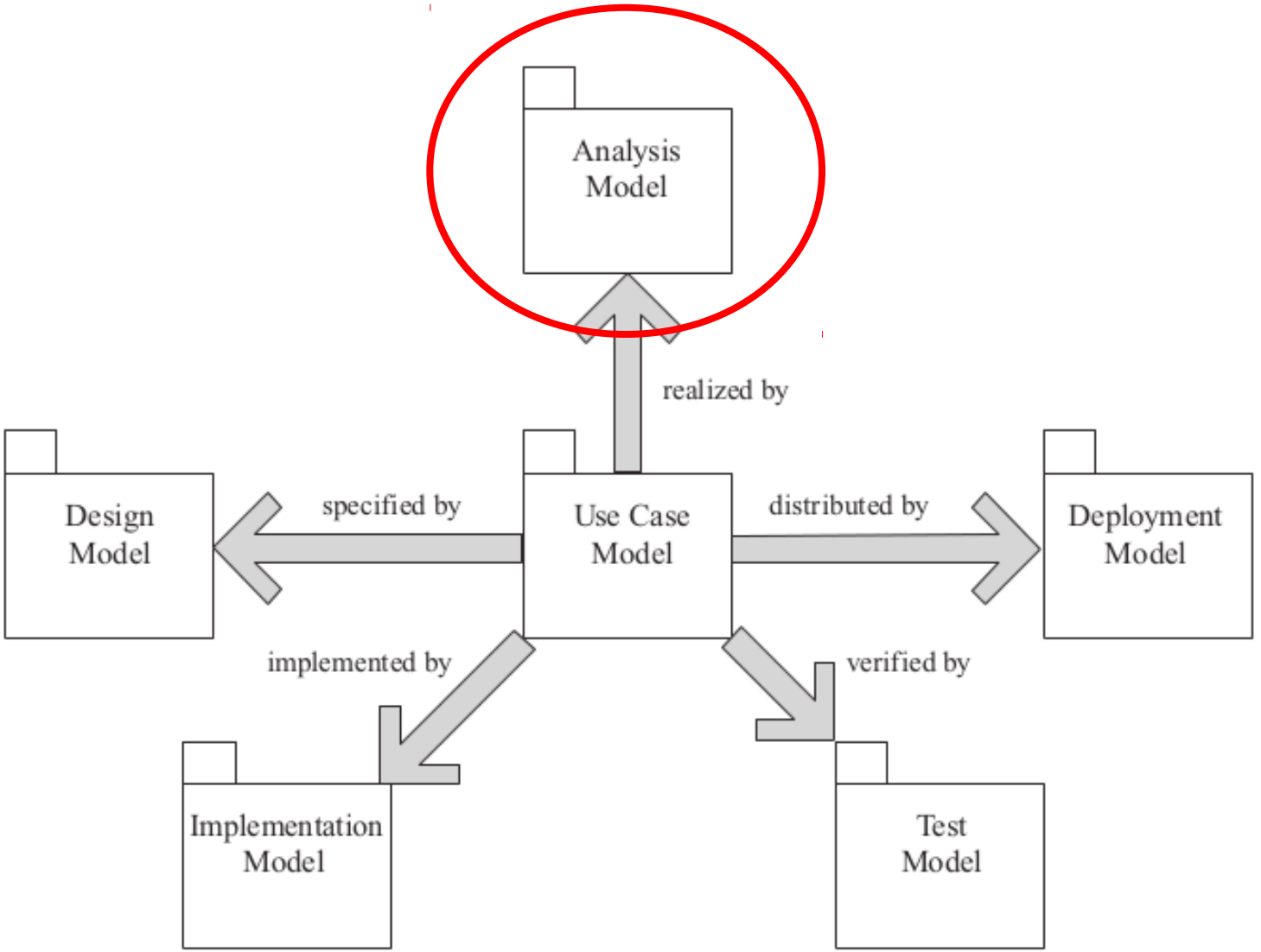
Les modèles dans UP



Modèle d'analyse

Dans UP, le modèle d'analyse décrit le système d'un point de vue structurel.

Il est construit à partir du modèle de cas d'utilisation.

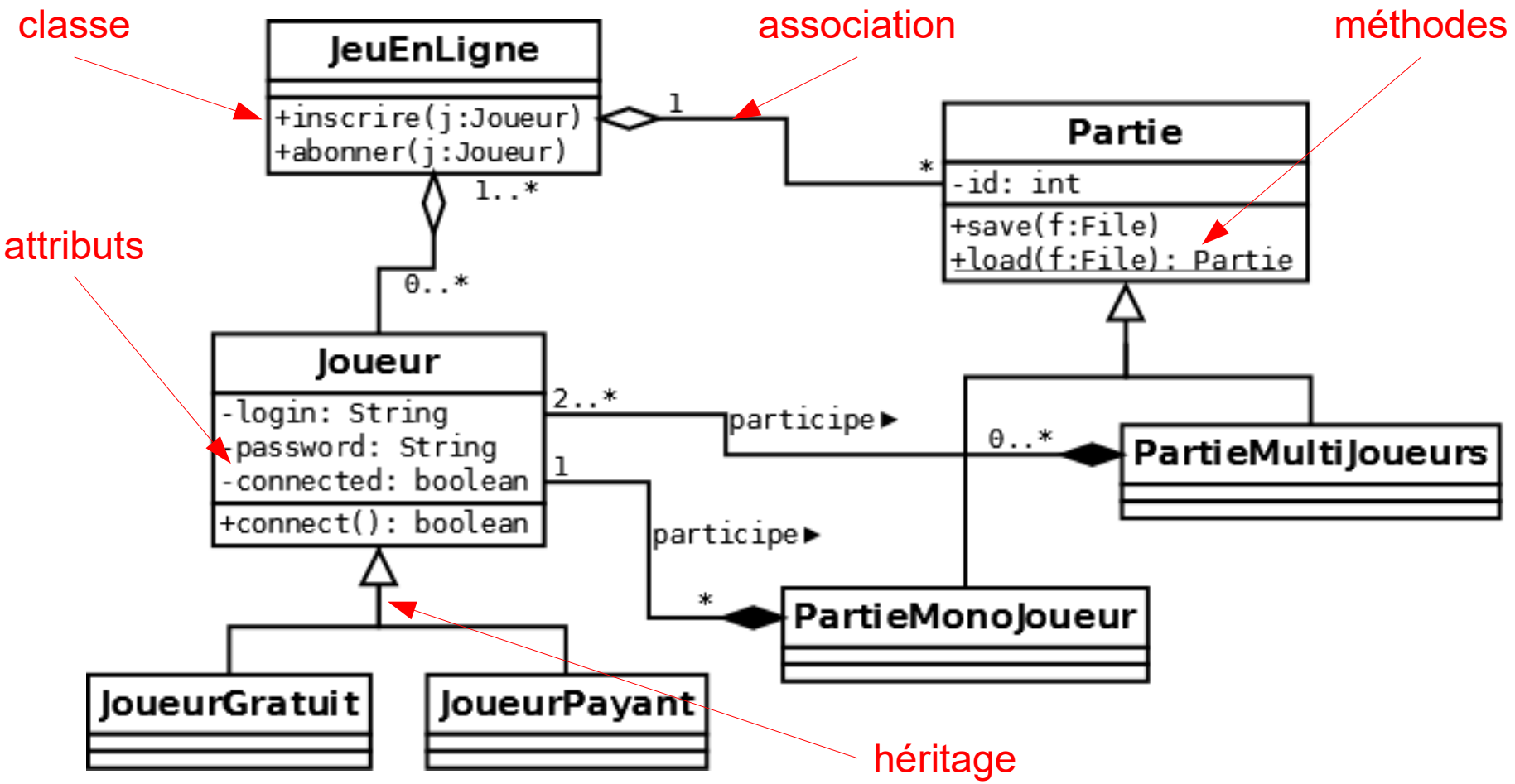
Il permet de définir l'architecture générale du système à l'aide principalement des diagrammes de classes et de paquetages, complétés avec des diagrammes d'objets et d'interaction.

Les classes n'ont pas à être forcément détaillées à ce stade (types des attributs, visibilité des membres, …).

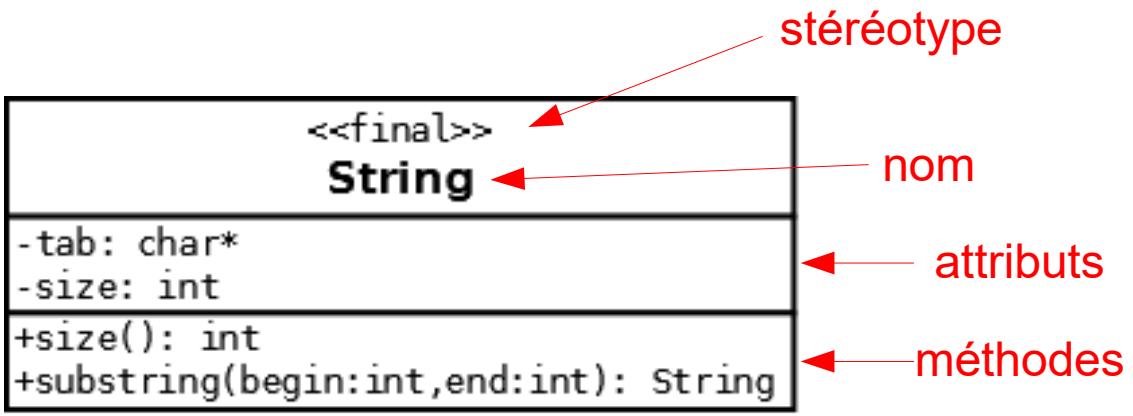
Le modèle de conception va compléter le modèle d'analyse en spécifiant les détails structurels et comportementaux.

Diagramme de classes

Un diagramme de classes décrit les classes et leurs relations (associations, généralisation/spécialisation, …).



Classe (1/2)



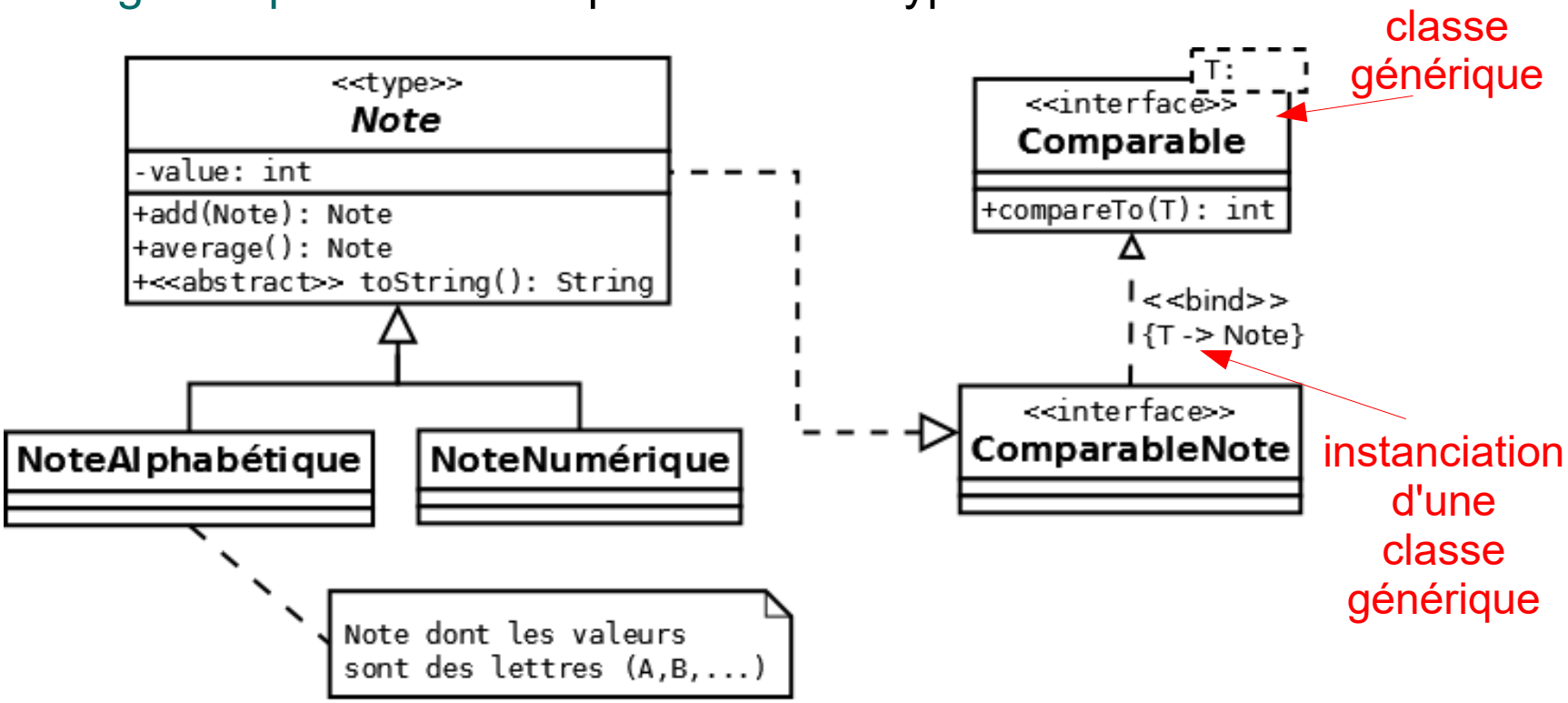
Une classe *abstraite* a son nom en italique (ou marquée avec le stéréotype `<<abstract>>`).

Le stéréotype `<<interface>>` est utilisé pour les interfaces.

Autres stéréotypes : `<<dataType>>` pour les types (classe dont les instances sont identifiés par leur valeur), `<<enumeration>>` pour les types énumérés, `<<utility>>` pour les classes contenant uniquement des attributs et méthodes statiques, `<<metaclass>>` pour les classes dont les instances sont des classes, ...

Classe (2/2)

Une classe générique donne son paramètre de type.



Une classe active est une classe qui contrôle son exécution :

Attributs

Format d'attribut : `[visibilité] ["/"] nom [multiplicité] [":" type] ["=" valeur initiale] ["{" propriétés "}"]`

`/` : indique que l'attribut est dérivé (redondant, calculé à partir d'autres)

Visibilité : `+` (public), `–` (privé), `#` (protégé), `~` (paquetage)

Multiplicité : `n` (exactement n instances), `n..m` (de n à m instances), a,b (exactement a ou exactement b instances)

Propriété : *`redefines`*, *`readOnly`*, *`frozen`*, *`ordered`*, ...

Un attribut statique est souligné

Exemples :
```
# / age : int = 0
~ notes[1..12] : int {ordered}
+ PI : float = 3.1415927 {frozen}
```

Méthodes

Format de méthode : `[visibilité] nom "(" [paramètres] ")" [":" type de retour]`

Visibilité : + (public), – (privé), # (protégé), ~ (paquetage)

Paramètre : `[mode] nom : type` mode = `in` (défault), `out` ou `inout`

Une méthode statique est soulignée, une méthode *abstraite* est en italique

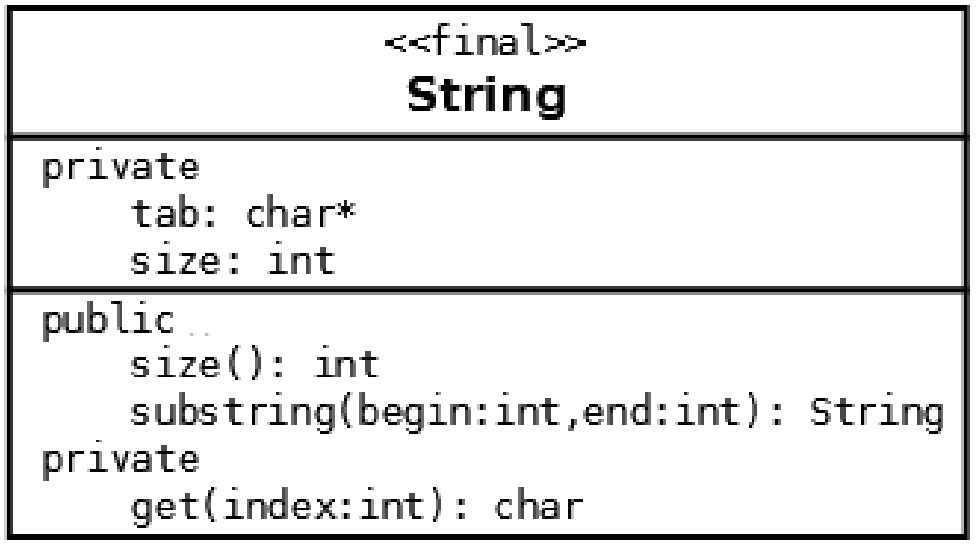
Un constructeur est nommé `create` et un destructeur `destroy`

Exemples :

```
+ trier (inout tableau : int[])
+ getName() : String
```

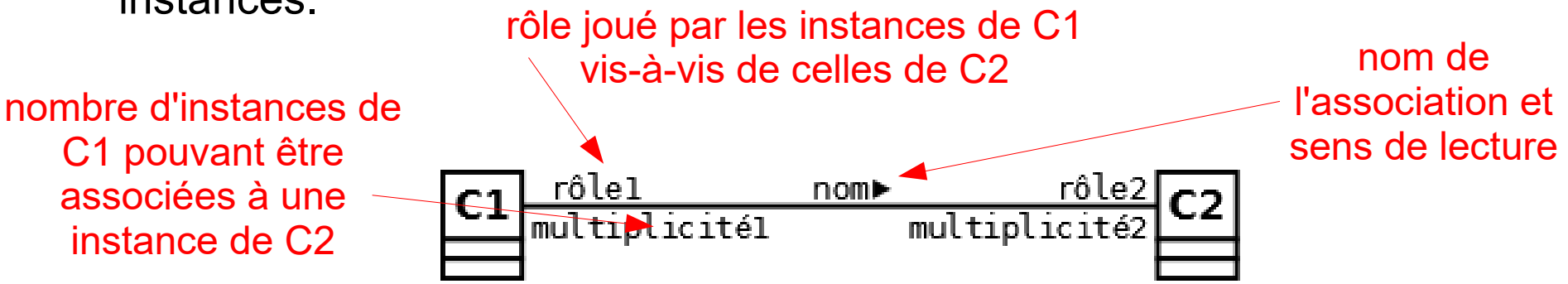
Visibilités dans une classe

Il est possible de regrouper les attributs et méthodes par visibilité.



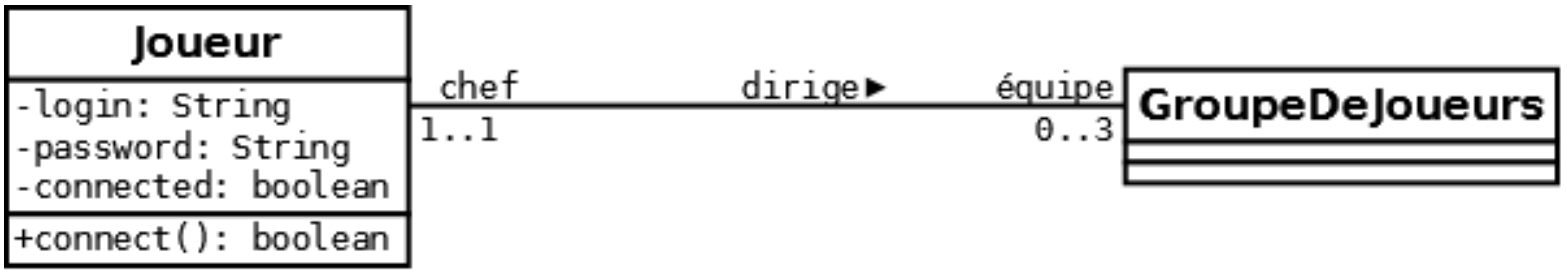
Associations (1/2)

Une association entre deux classes indique qu'un lien existe entre leurs instances.



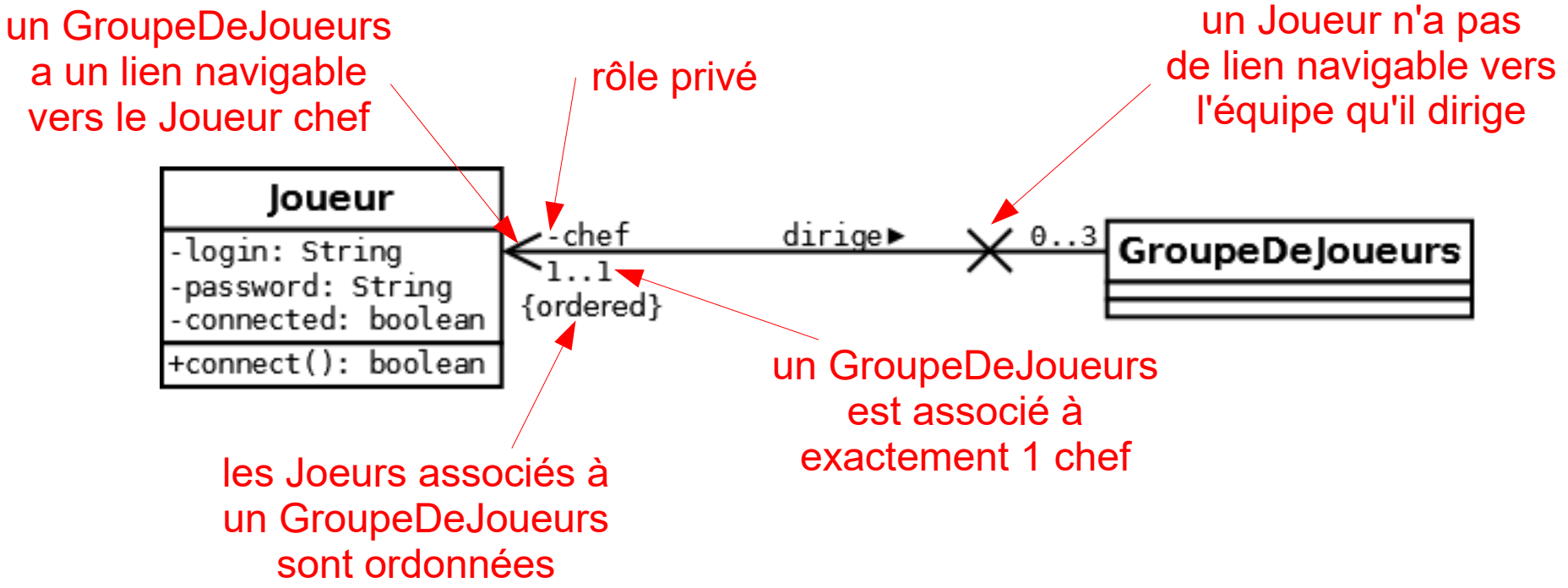
Multiplicité : *, 1, 0..1, 1..1, 0..*, 1..*, 3..7, …

Exemple : un joueur dirige un groupe de joueurs, il joue le rôle de chef pour une équipe. Un groupe a exactement un chef et un joueur peut diriger au plus 3 équipes.



Associations (2/2)

Il est possible de spécifier la navigabilité d'une association et la visibilité des rôles.

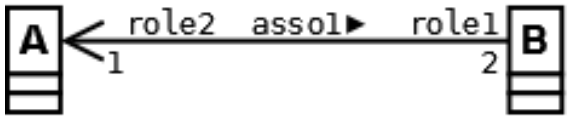


Propriétés sur les extrémités : `addOnly`, `frozen`, `ordered`, ...

Implémentations des associations (1/2)

Une association est une entité en elle-même et possède ses terminaisons (instances).

On pourra par exemple implémenter une association par une classe (cf. classe-association).




```
class asso1{
    A monA;
    B monB1, monB2;
}
```

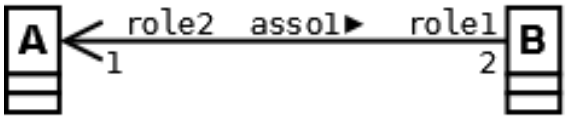
On peut aussi implémenter une association par des méthodes : les terminaisons appartiennent alors aux classes.

```
class B{
    ...
    A getRole2(){
        ...
    }
}
```

```
class A{
    ...
    B[] getRole1(){
        ...
    }
}
```

Implémentations des associations (2/2)

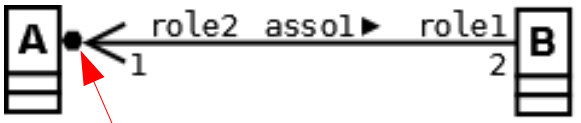
Le plus souvent, on implémente une association par des attributs.




```
class B{
    A role2;
}
```

```
class A{
    B role1_1, role1_2;
}
```

On peut spécifier l'implémentation en imposant que les terminaisons de l'association appartiennent aux classes.



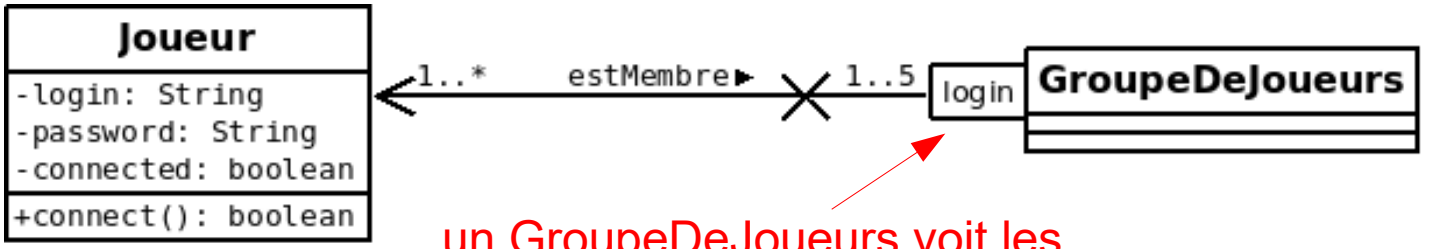
cette extrémité de l'association appartient à la classe B : les instances de B doivent avoir une référence sur 1 instance de A

```
class B{
    A role2;
}
```

```
class asso1A2B{
    A monA;
    B monB1, monB2;
}
```

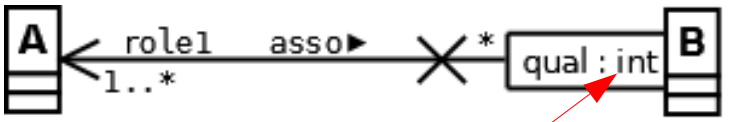
Association qualifiée

Une association qualifiée lie deux classes en spécifiant une valeur de type clé ou index qui qualifie les objets liés.



un GroupeDeJoueurs voit les Joueurs auxquels il est associé via l'attribut login

Une association qualifiée est généralement implémentée par une structure de type tableau associatif.

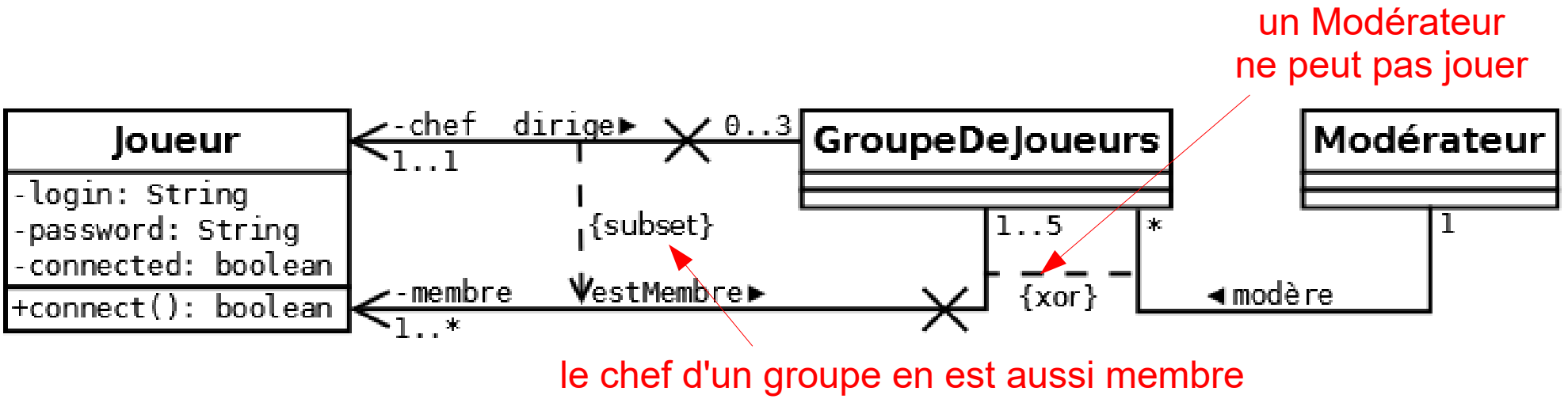


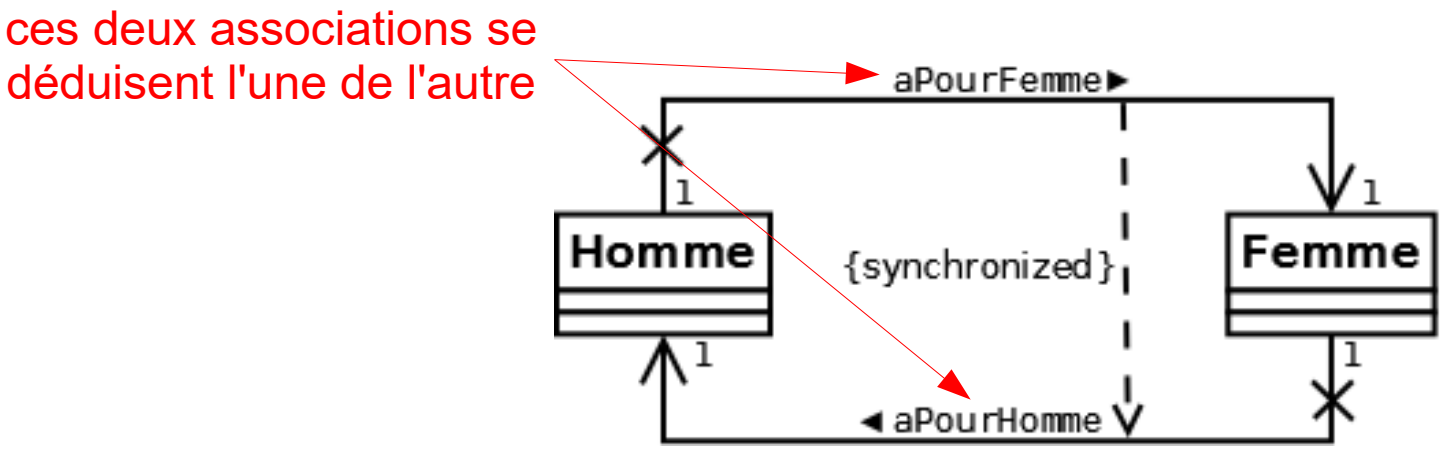
on peut spécifier le type de l'attribut qualifiant

```
class B{
    Hashtable<Integer,A> mesA;
    ...
    void addRole1(int qual, A a){...}
    A getRole1(int qual){...}
}
```

Relations entre associations

Des relations conceptuelles peuvent être spécifiées entre associations.

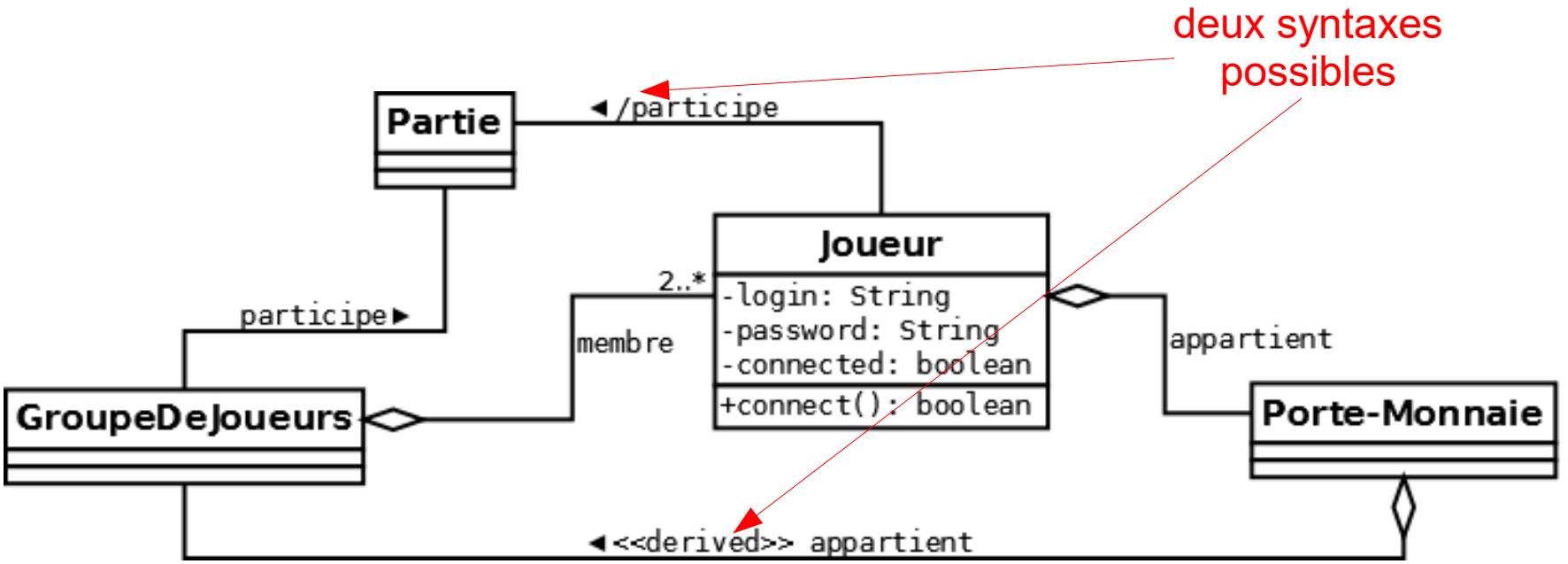




Associations dérivées

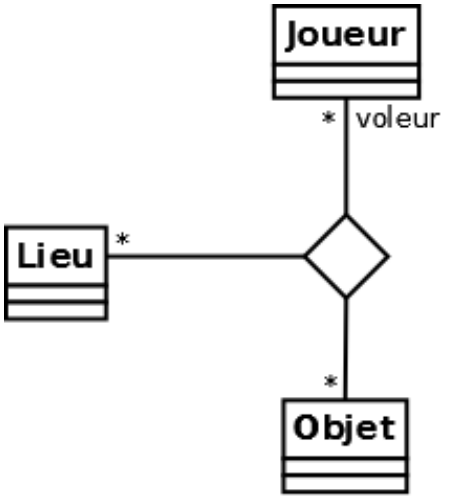
Il est possible de spécifier qu'une association est dérivée c'est-à-dire déduite d'autres.

Les associations dérivées peuvent être spécifiées pour améliorer la lisibilité du diagramme.

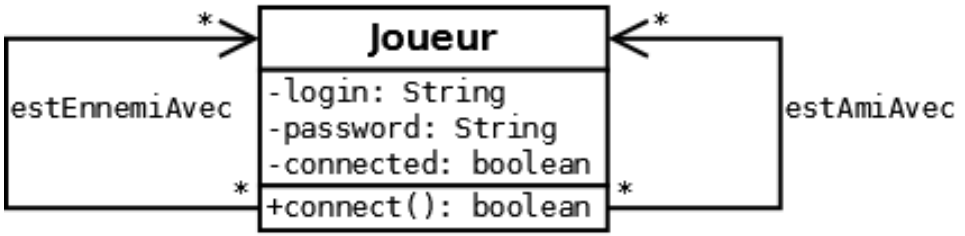


Types d'associations

Une association n-aire permet d'associer plus de 2 classes (peu lisible, à éviter) :

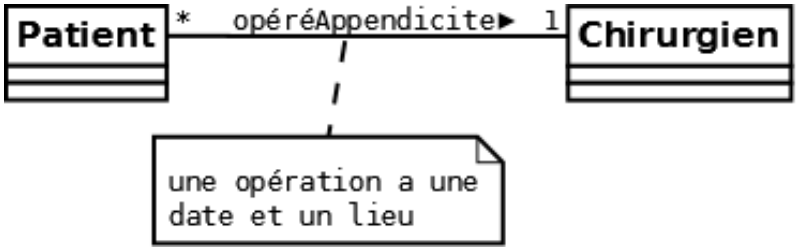


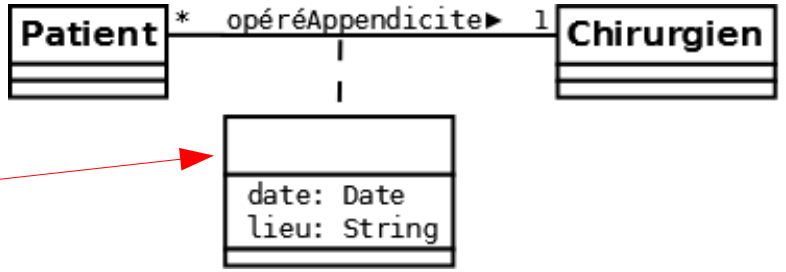
Une association réflexive associe une classe à elle-même :



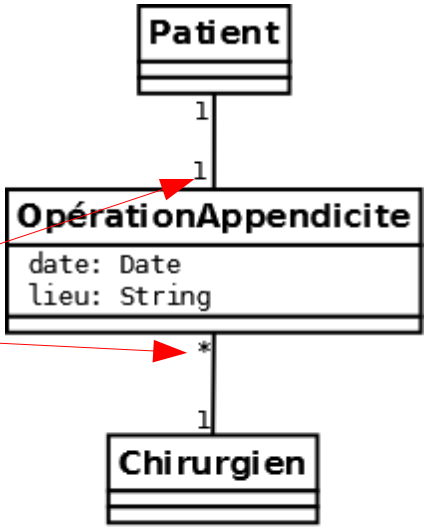
Classe-association

Une association décrite par des propriétés doit être implémentée par une classe et est modélisée par une classe-association :





classe-association (éventuellement anonyme)

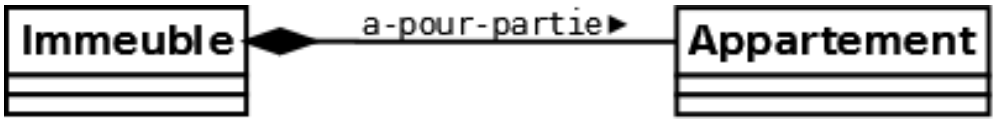


les cardinalités se retrouvent sur la classe association et les cardinalités sur les classes extrémités sont forcément à 1

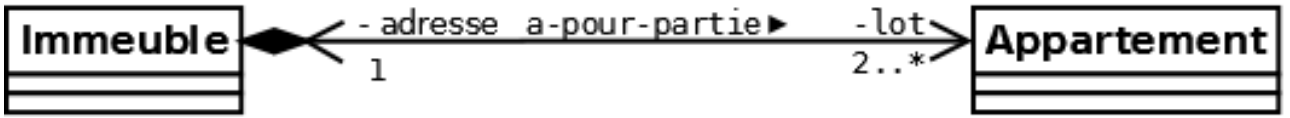
Composition et agrégation (1/3)

Une composition est une association de type composant/composite ou tout-partie :

- transitive et antisymétrique
- la création ou destruction du composite entraîne celle des composants
- un composant ne peut appartenir qu'à un composite

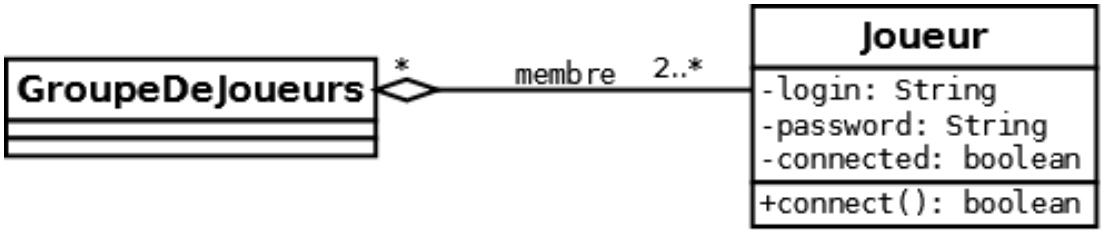


La composition n'empêche pas de spécifier les rôles, la navigabilité et la multiplicité.

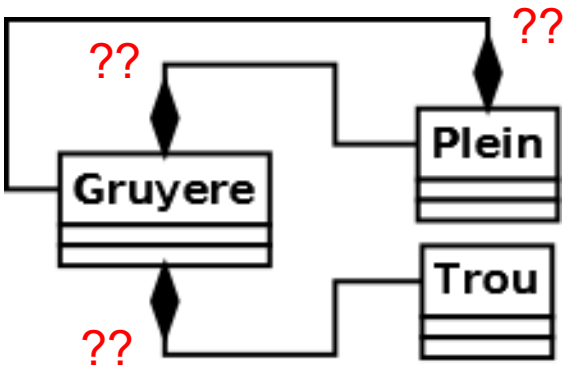


Composition et agrégation (2/3)

Dans le cas d'une agrégation, les composants ont une existence indépendante de celle du composite et peuvent appartenir à plusieurs composites :

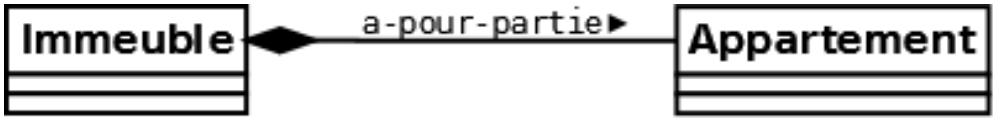


Une double composition ou agrégation n'a pas de sens puisque ces relations sont antisymétriques :



Composition et agrégation (3/3)

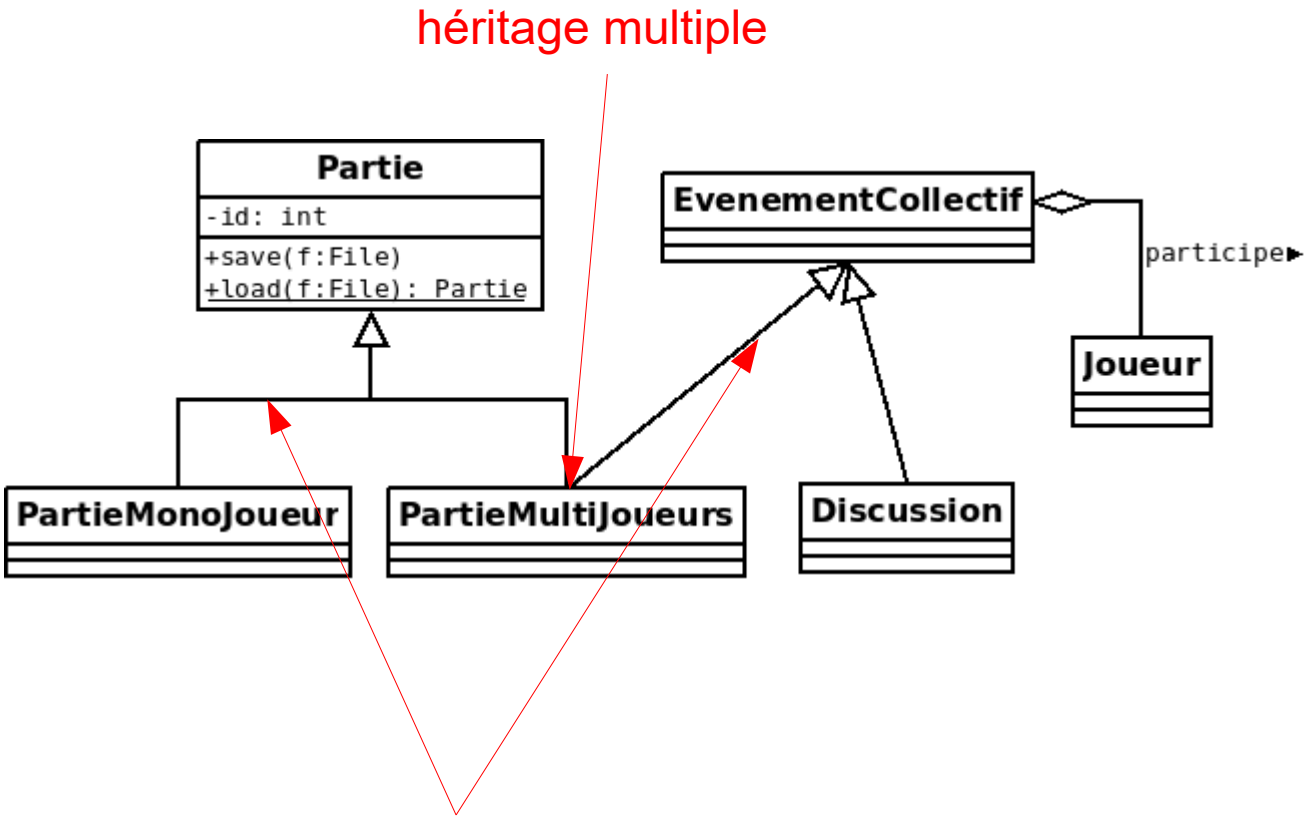
Une relation de méronymie (partie-tout) s'implémente comme une association normale, mais dans le cas d'une composition, la destruction du tout doit entraîner la destruction des parties.




```
class Immeuble{
    std::list<Appartement> contenu;
    ...
    ~Immeuble(){
        // prévoir ici la destruction des appartements
        ...
    }
}
```

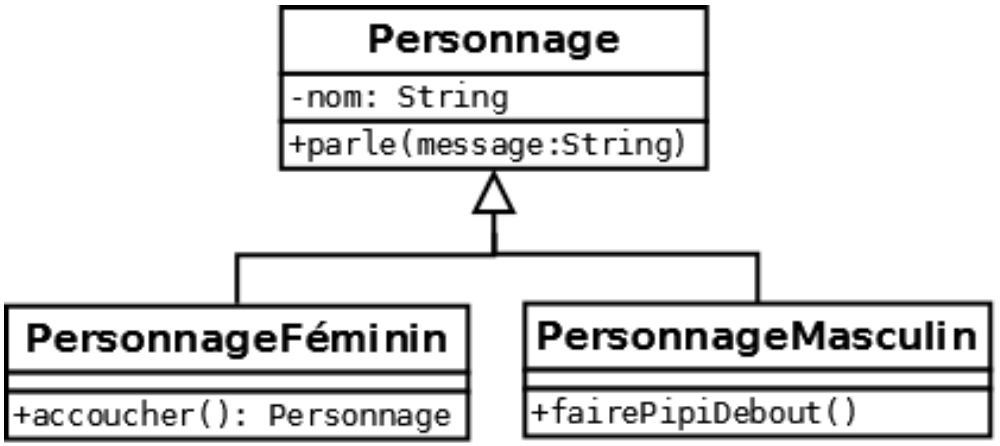
Généralisation de classe

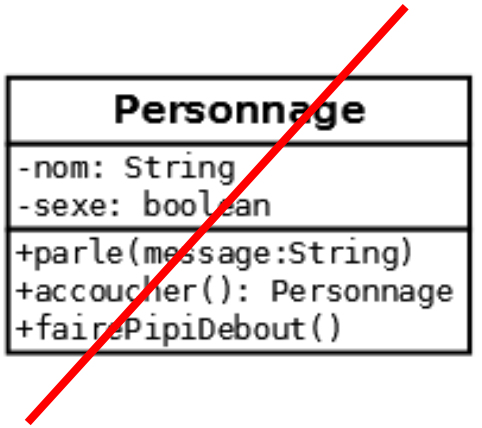
La généralisation correspond à la notion d'héritage.



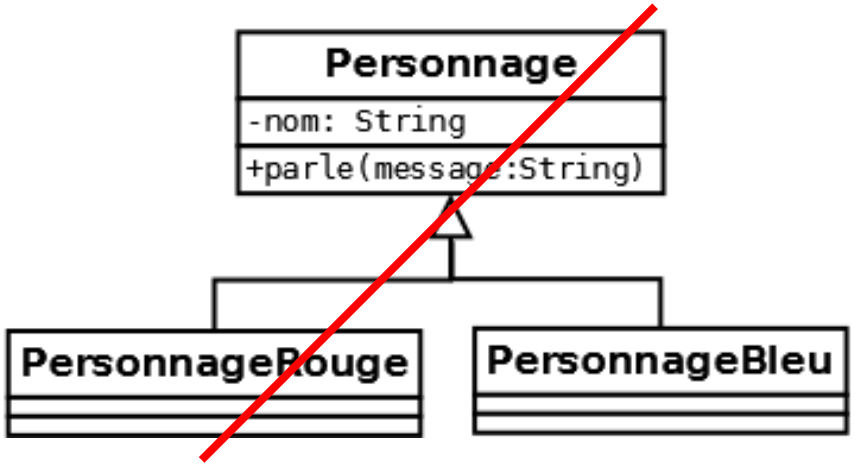
Généralisation où composition ? (1/2)

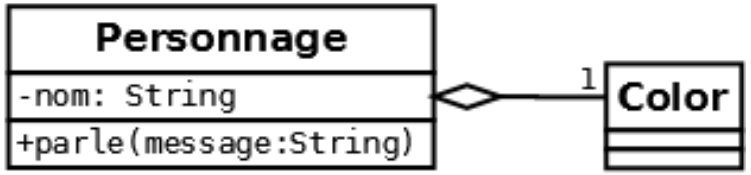
Exemple : un personnage a un nom et un sexe, peut parler, peut accoucher si c'est une femme et faire pipi debout si c'est un homme.





Exemple : un personnage peut parler, peut être rouge ou bleu.



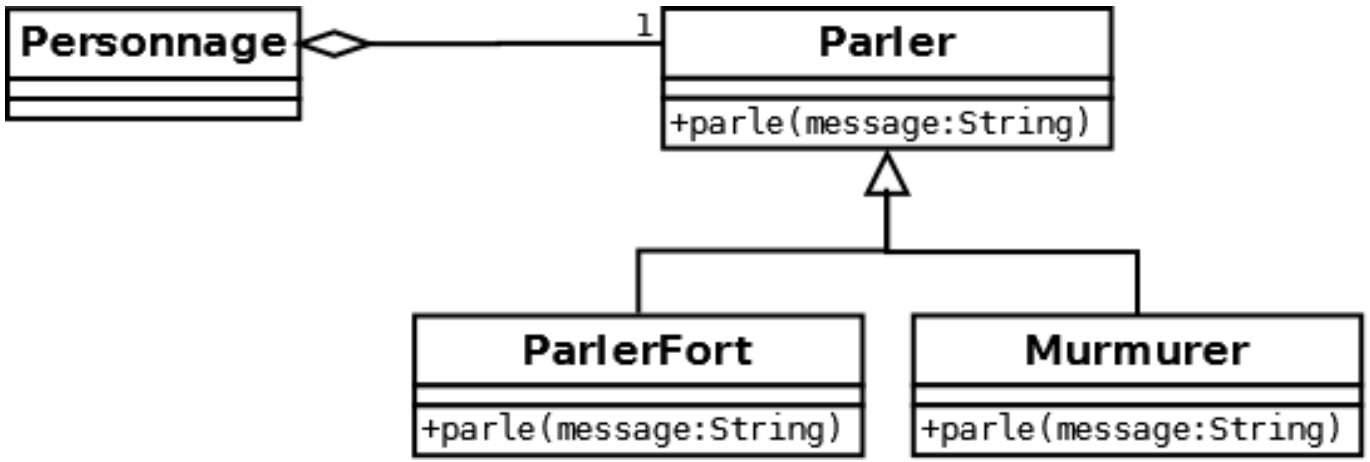


Généralisation où composition ? (2/2)

La composition peut être préférable à la généralisation, car elle est plus résistante aux changements.

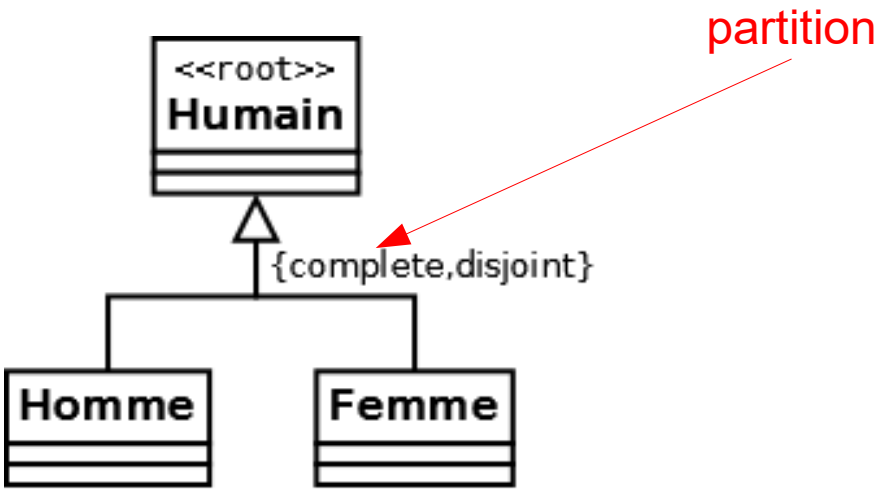
Exemple : finalement, les personnages peuvent aussi être verts

La composition peut même servir à attribuer des comportements différents à des instances de la même classe (délégation).



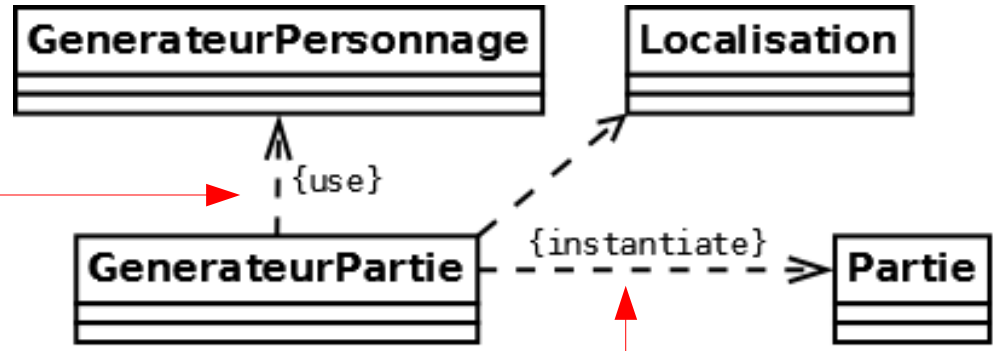
Stéréotypes de généralisation

Les liens de généralisation peuvent porter des propriétés : `{disjoint}`, `{complete}`, `{leaf}`, `{root}`, ...



Dépendance entre classes

Des liens de dépendance peuvent exister entre classes. Souvent on les précise par des stéréotypes.

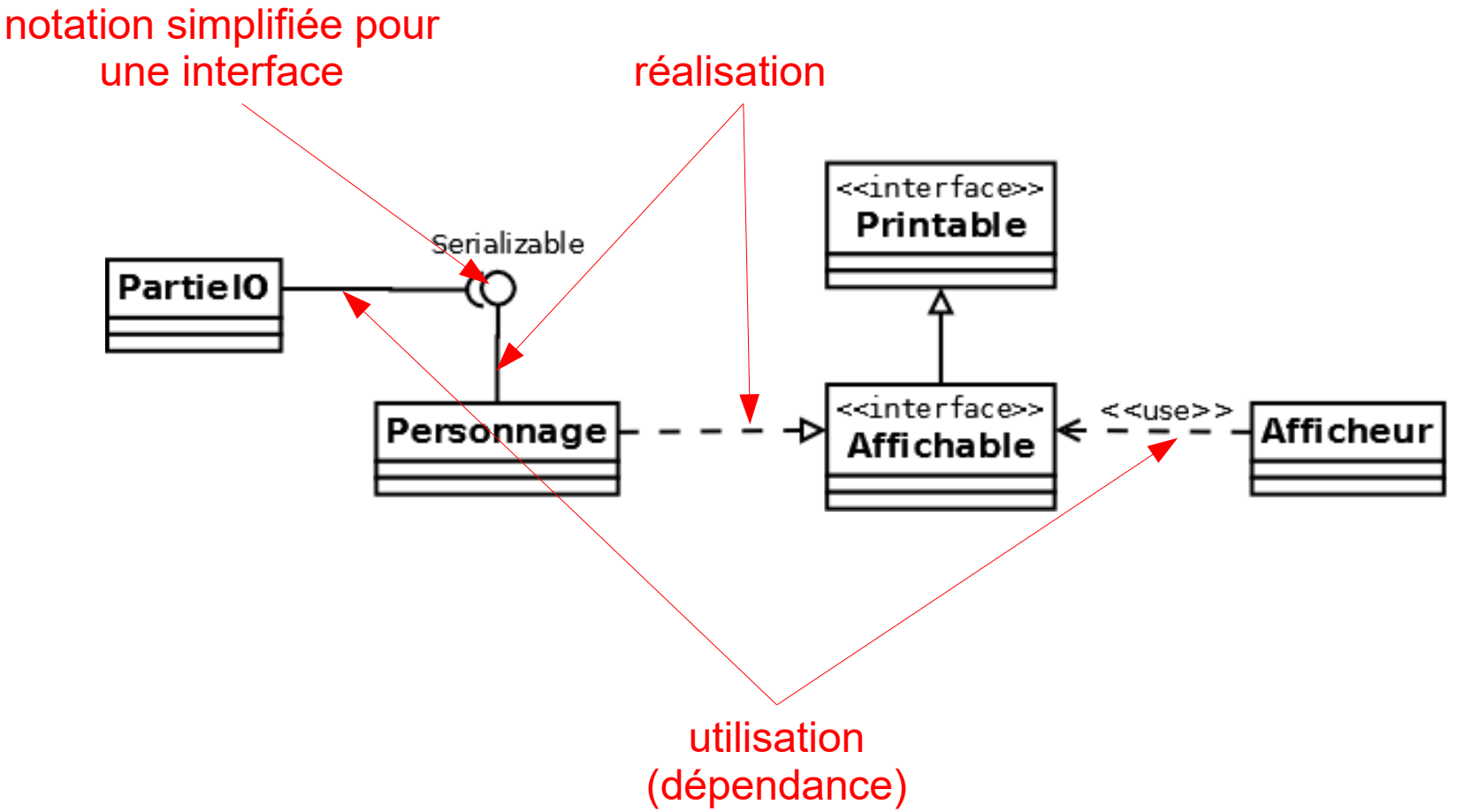


le générateur de partie fait appel au générateur de personnage

le générateur de partie crée des instances de Partie

Interfaces

Les interfaces sont spécifiées par un stéréotype et peuvent être réalisées (implémentées) ou utilisées par des classes.



Classes internes

Les classes internes sont spécifiées par un type de lien particulier avec leur classe englobante.

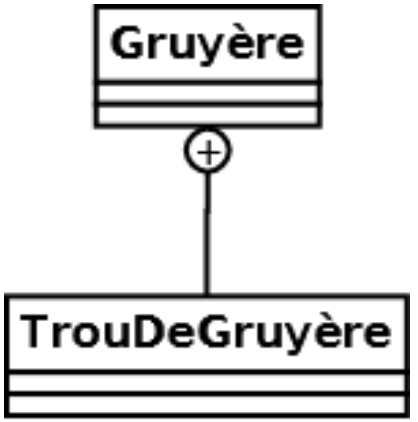


Diagramme d'objets (1/2)

Un diagramme d'objets décrit l'état des objets dans le système en train de s'exécuter à un moment donné.

Les liens entre objets sont de même type que les associations entre classes.

Il permet de représenter un cas de test, mais aussi d'analyser des exemples d'exécution.

le type d'un attribut peut être omis

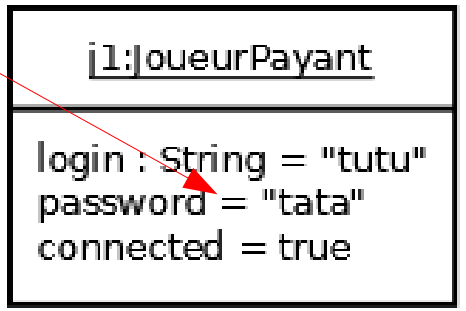
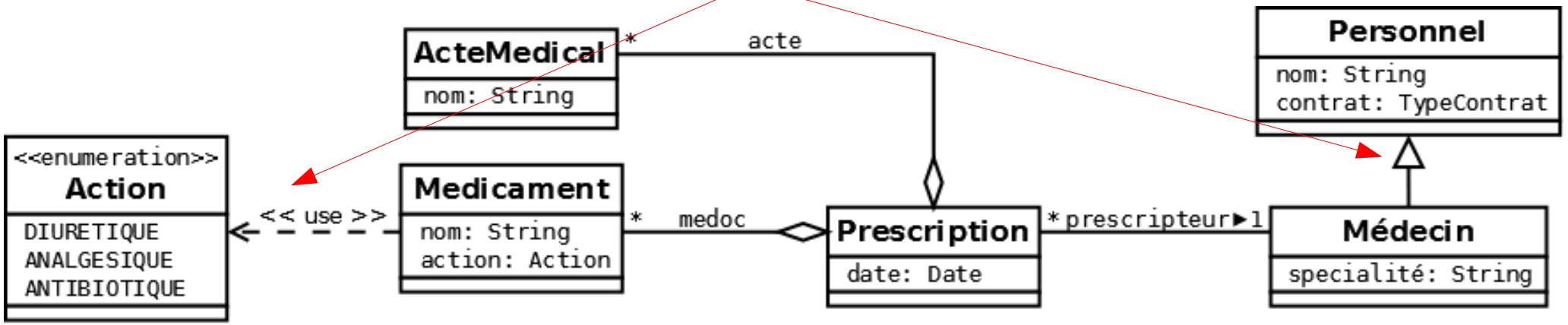


Diagramme d'objets (2/2)

tous les éléments du diagramme de classe ne se retrouvent pas forcément dans un diagramme d'objet



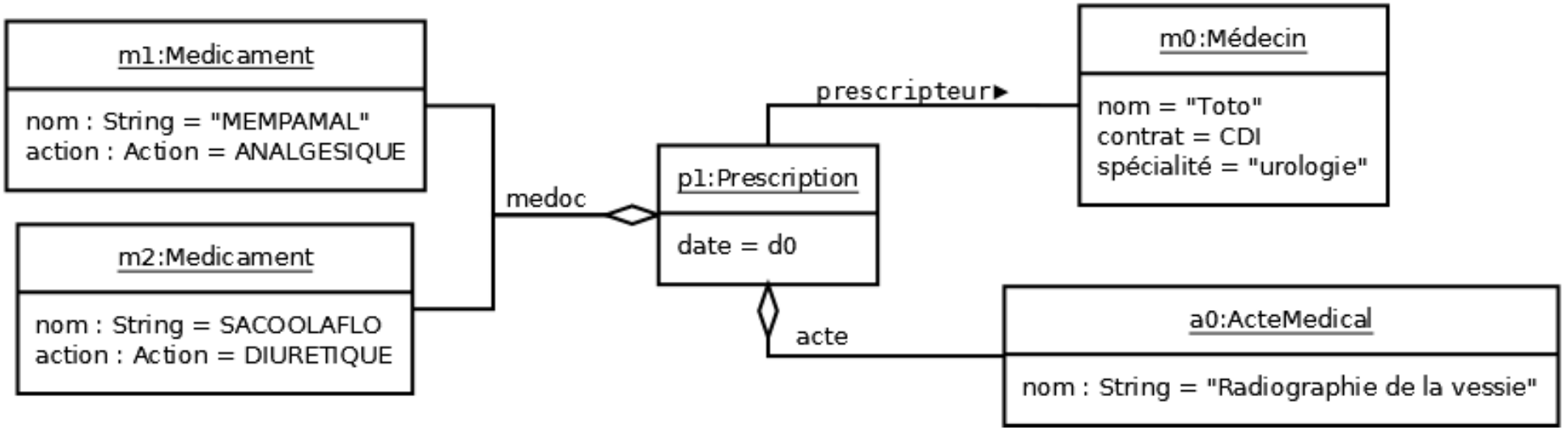


Diagramme de timing (1/2)

Le diagramme de timing est un type particulier de diagramme d'interaction qui permet de décrire la vie d'un objet avec des contraintes temporelles.

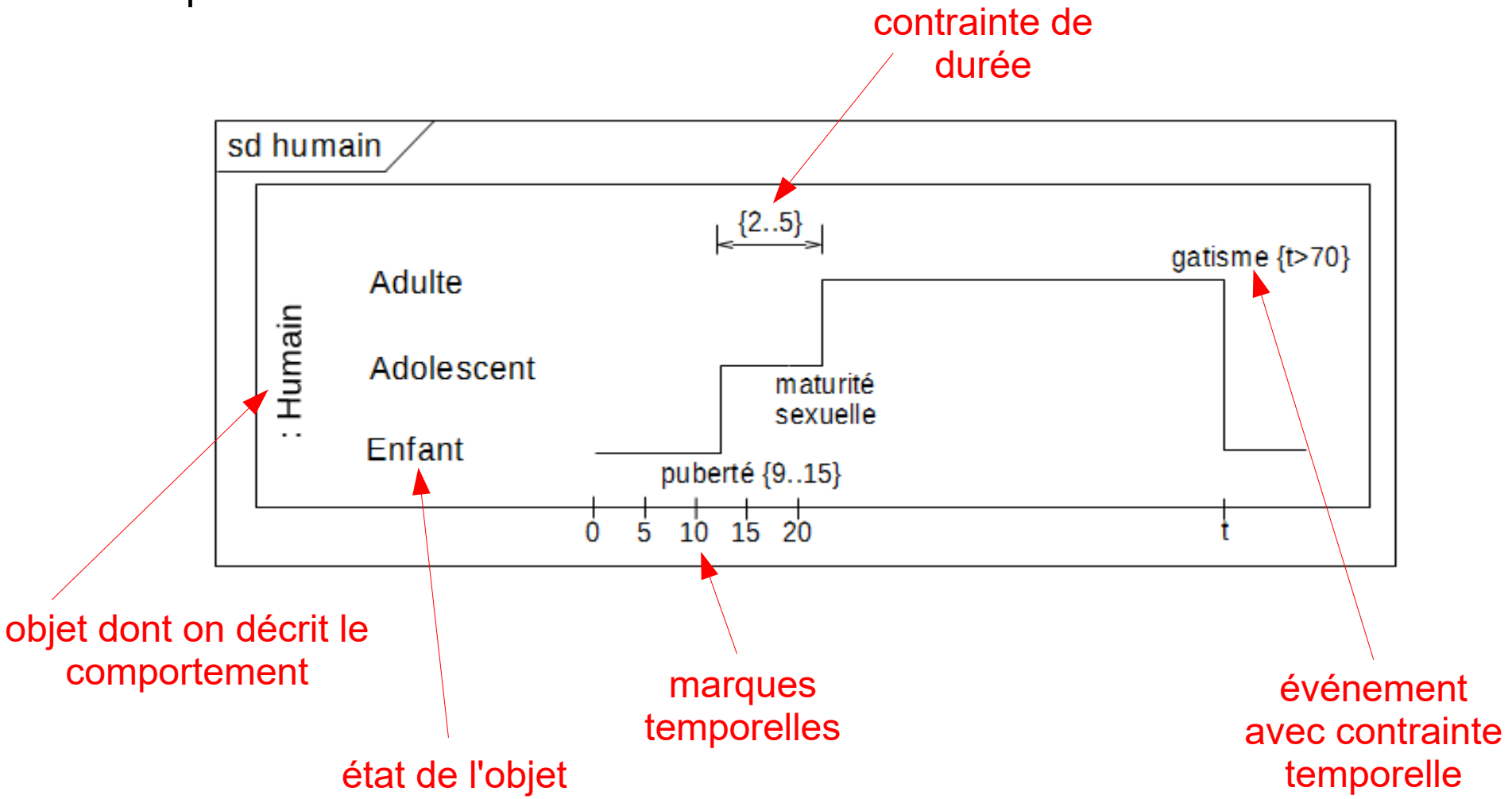
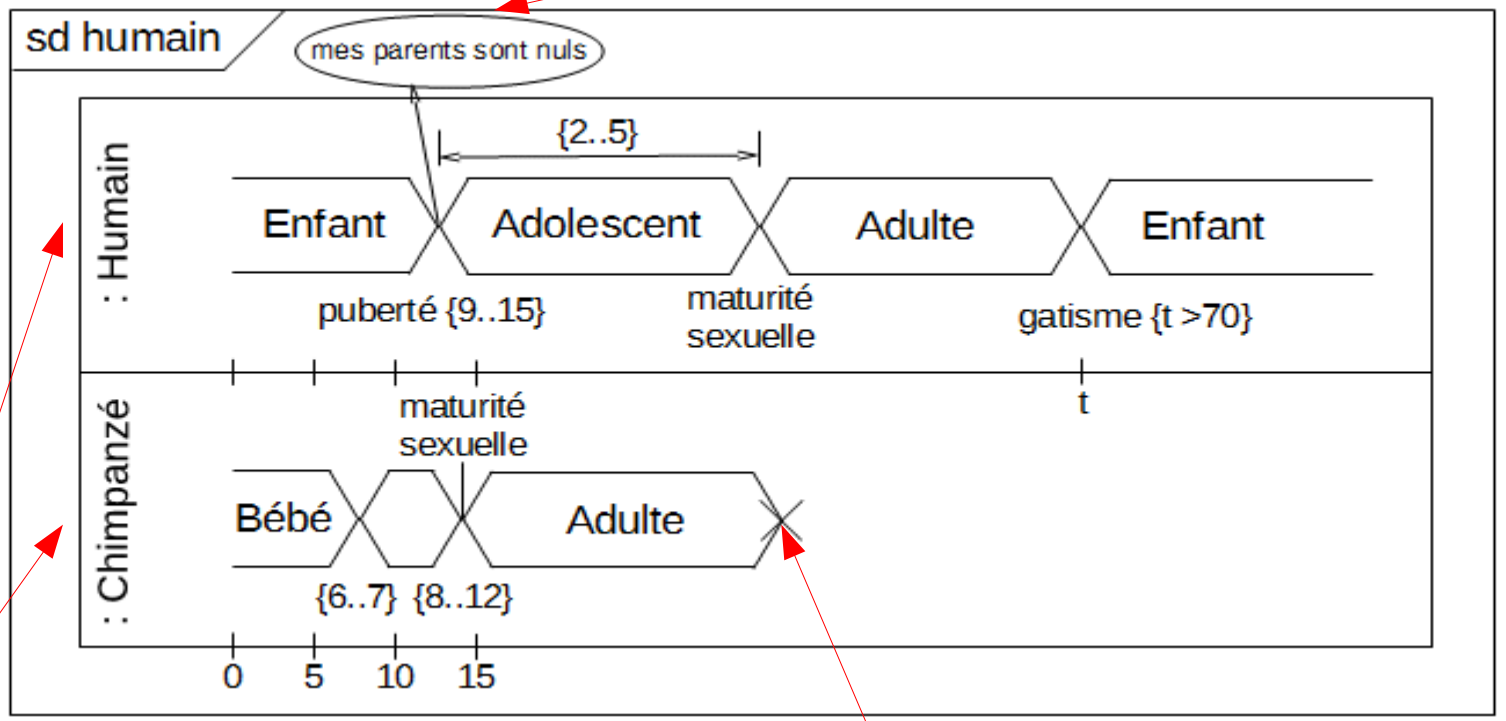


Diagramme de timing (2/2)

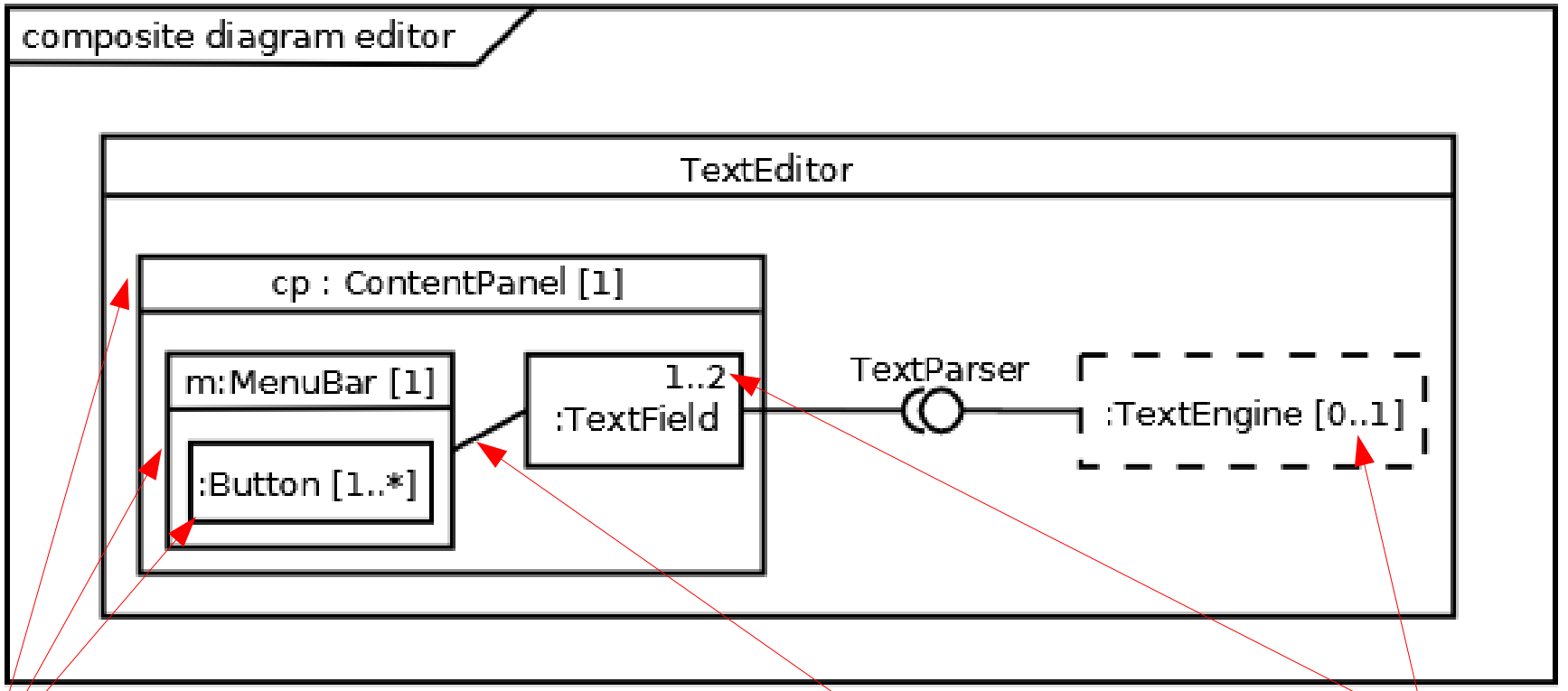
Une syntaxe simplifiée est possible :



Structure composée

Le diagramme de structure composée (composite structure) permet de visualiser la structure interne d'une classe.

On distingue les compositions ou parties (traits pleins) et les autres associations ou rôles (pointillés).



l'imbrication peut être sur plusieurs niveaux

connecteur

multiplicité
(deux syntaxes possibles)

Connecteurs et ports

Un connecteur représente un lien entre instances. Un connecteur peut être une instance d'association ou implémenté par un pointeur, un passage de paramètre, un autre objet, un mécanisme de communication réseau, etc.

Un port représente un point d'interaction entre la classe et son environnement logiciel.

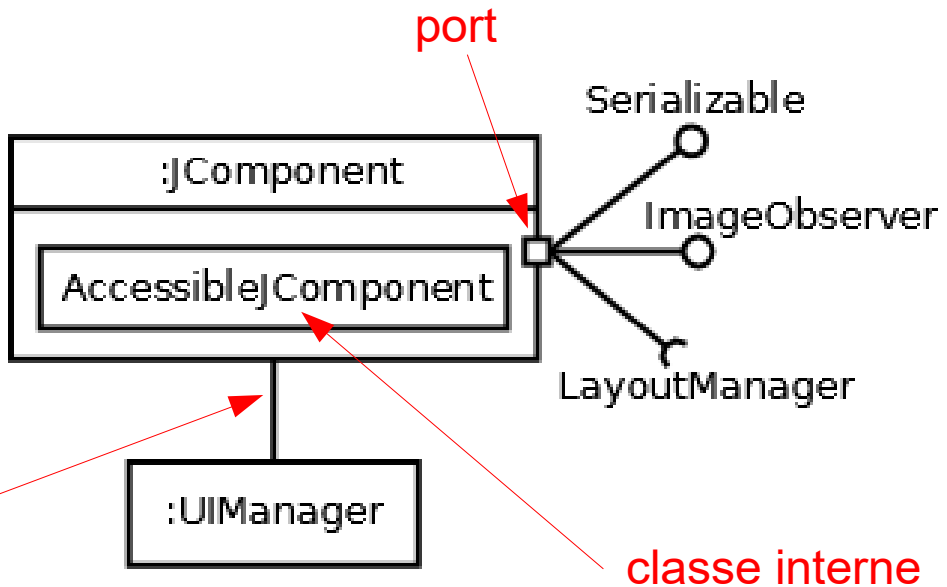


Diagramme de paquetages (1/3)

Un diagramme de paquetage décrit à grande échelle l'architecture du système.

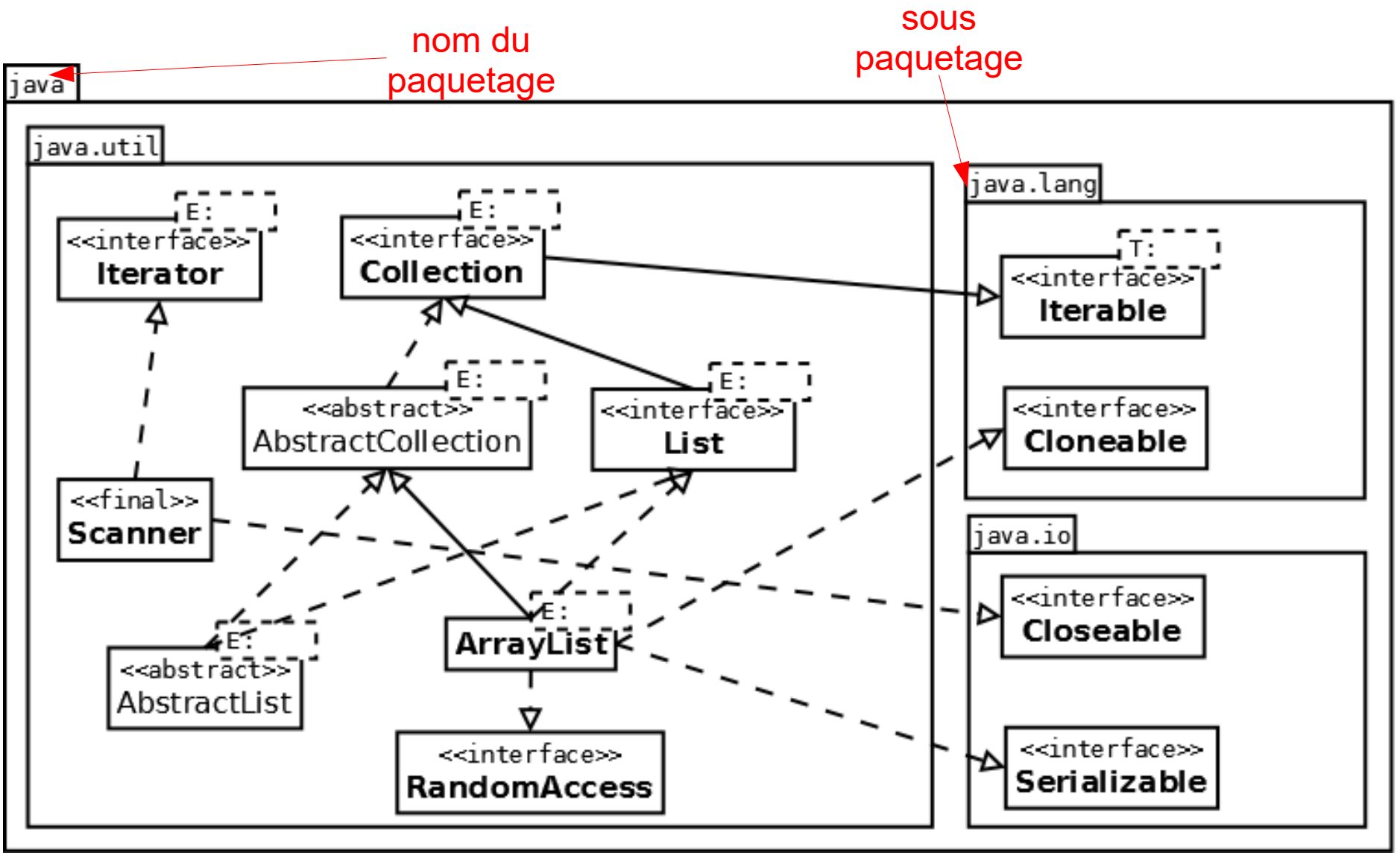


Diagramme de paquetages (2/3)

Autre syntaxe possible :

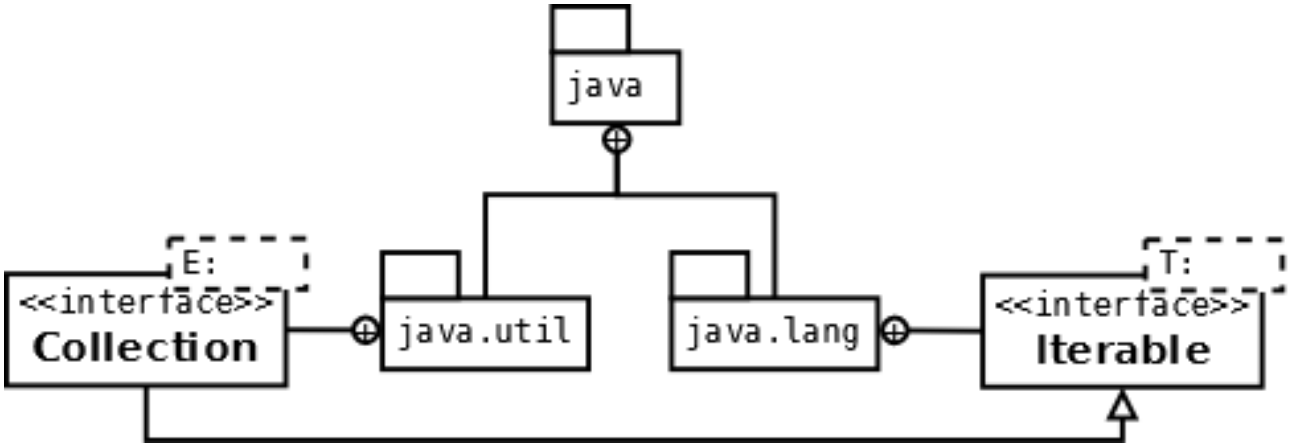
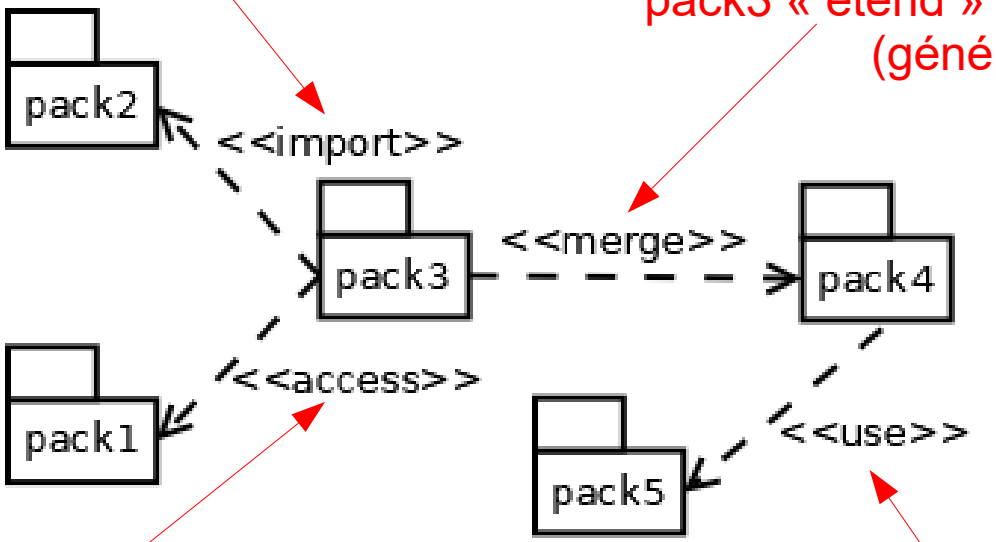


Diagramme de paquetages (3/3)

Stéréotypes sur les dépendances

des classes de pack3 importent des classes de pack2

pack3 « étend » le contenu de pack4 (généralisation)



des éléments de pack3 utilisent des éléments de pack1 (sans forcément importer des classes)

pack4 dépend de pack5