# Chapitre 3 :
# Object Constraint Language (OCL)

# C'est quoi OCL?



- **Pourquoi OCL ? Introduction par l'exemple**
- Les principaux concepts d'OCL
- Exemple d'application
- Complément OCL

Ce cours est basé sur un cours de
Eric CARIOU https://ecariou.perso.univ-pau.fr/cours/spec.html

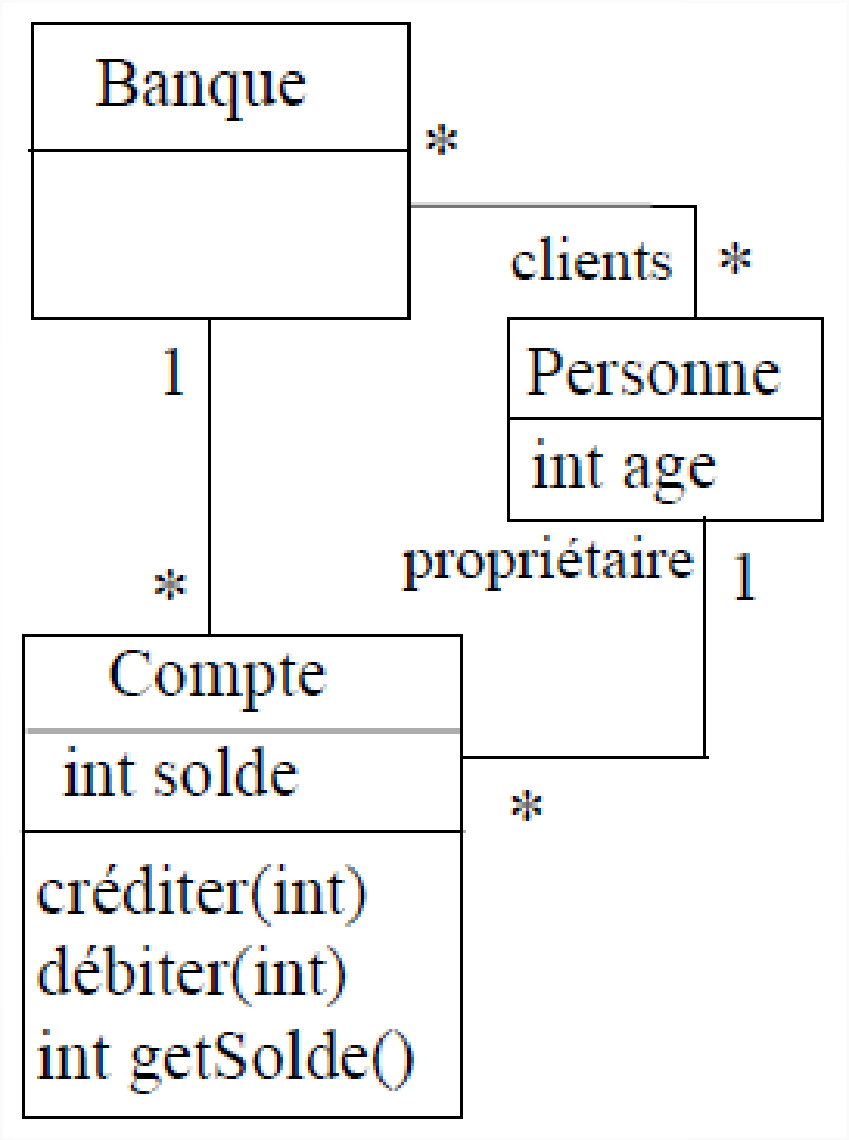
# Exemple d'application



- Application bancaire :
  - Des comptes bancaires
  - Des clients
  - Des banques
- Spécification :
  - Un client peut posséder plusieurs comptes
  - Un client peut être client de plusieurs banques
  - Une banque gère plusieurs comptes
  - Une banque possède plusieurs clients
  - **Un compte doit avoir un solde toujours positif**
  - **Un client d'une banque possède au moins un compte dans cette banque**

# Exemple d'application





# Manque de précision

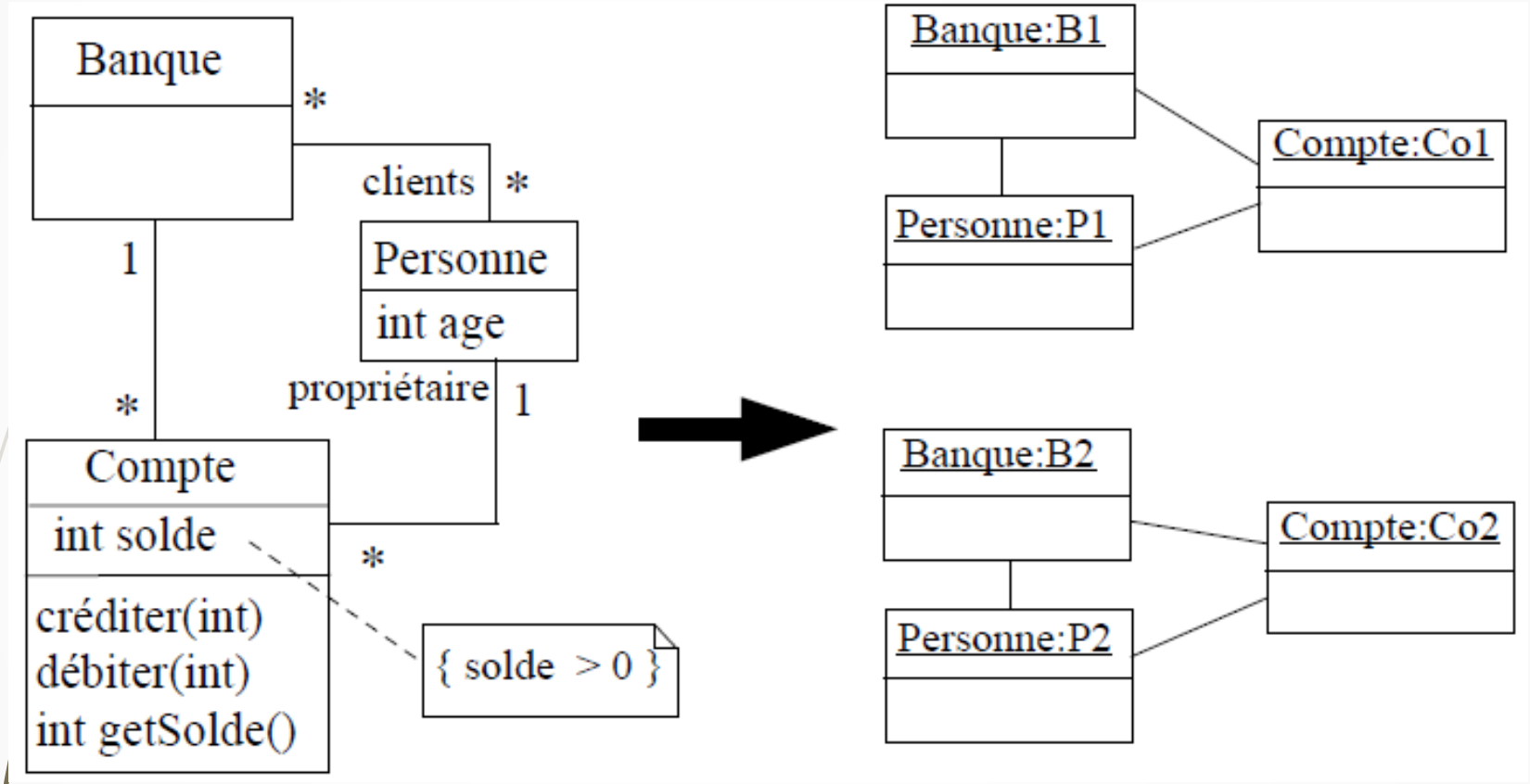


- Le diagramme de classe ne permet pas d'exprimer tout ce qui est défini dans la spécification informelle

- Exemple :
  - Le solde d'un compte doit toujours être positif ⇒ ajout d'une contrainte sur cet attribut
  - Le diagramme de classe ne permet pas de détailler toutes les contraintes sur les relations entre les classes

# Diagramme d'objets

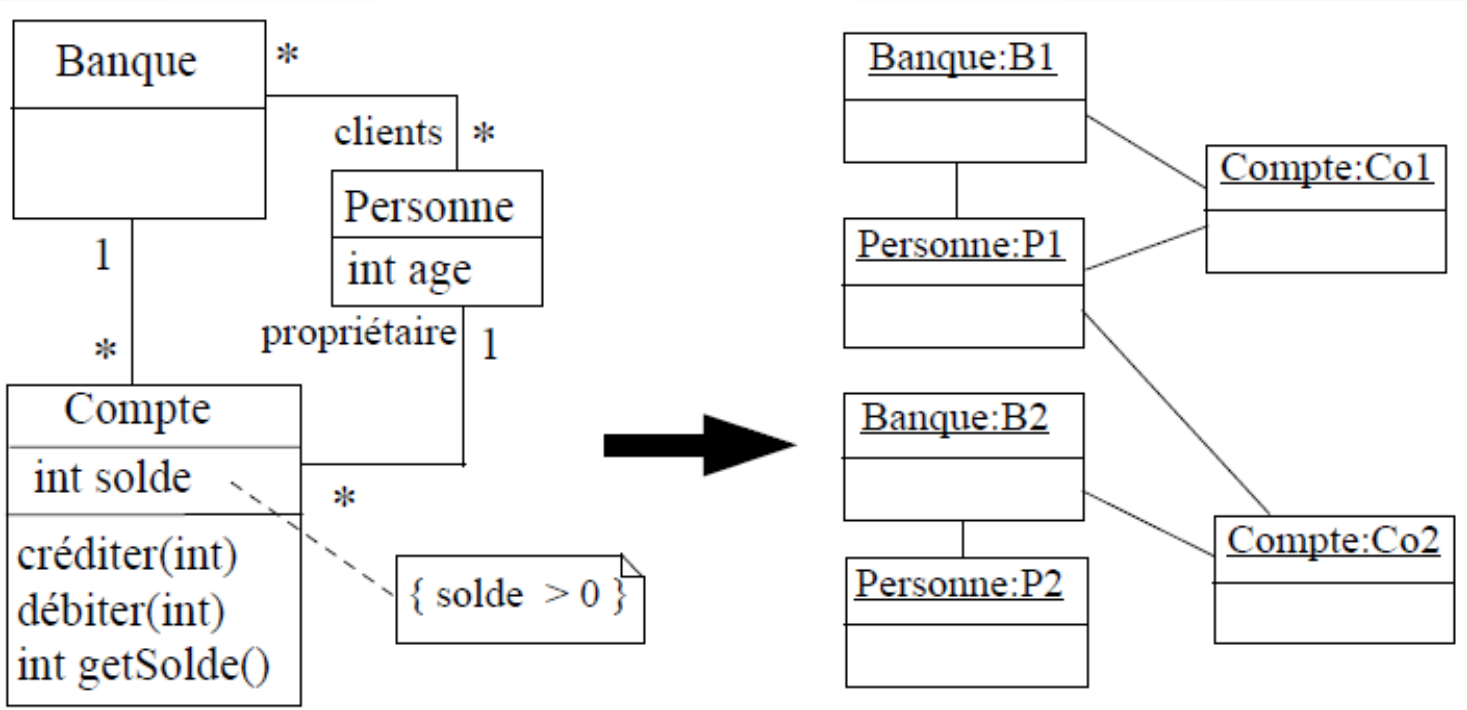





- Diagramme d'objets valide vis-à-vis du diagramme de classe et de la spécification attendue

# Diagramme d'objets






- Diagramme d'objets **valide** vis-à-vis du diagramme de classe **mais ne respecte pas la spécification attendue** :
- **Un client d'une banque possède au moins un compte dans cette banque**
- En effet:
  - P1 a un compte dans une banque (B2) où elle n'est pas cliente
  - P2 est cliente d'une banque (B2) mais sans y avoir de compte

# Diagrammes UML insuffisants



- Pour spécifier complètement une application :
  - Diagrammes UML seuls sont généralement insuffisants
  - Nécessité de rajouter des contraintes
- Comment exprimer ces contraintes ?
  - Langue naturelle mais manque de précision, compréhension
    - pouvant être ambigüe
  - Langage formel avec sémantique précise : par exemple OCL
- OCL : Object Constraint Language
  - Langage de contraintes orienté-objet
  - Langage formel (mais simple à utiliser) avec une syntaxe, une grammaire, une sémantique.

# C'est quoi OCL?



- Pourquoi OCL ? Introduction par l'exemple
- **Les principaux concepts d'OCL**
- Exemple d'application
- Complément OCL

# C'est quoi OCL?



- Un langage pour exprimer les contraintes dans les modèles UML.
- Un langage formel (sans ambiguïté) qui peut être lu et compris par les différents développeurs et clients
- Langage déclaratif : il décrit le quoi mais pas le comment
- Dernière version : OCL 2.4

# Le langage OCL



- OCL fait partie de la norme UML 1.X de l'OMG (Object Management Group)
- OCL en version 2.0 : spécification à part de la norme UML 2.0, normalisée par l'OMG
  - Peut s'appliquer sur tout type de modèle
- OCL permet principalement d'exprimer trois types de contraintes sur l'état d'un objet ou d'un ensemble d'objets.
- Attention : une expression OCL décrit une contrainte à respecter et pas le « code » d'une méthode

# C'est quoi une contrainte OCL?

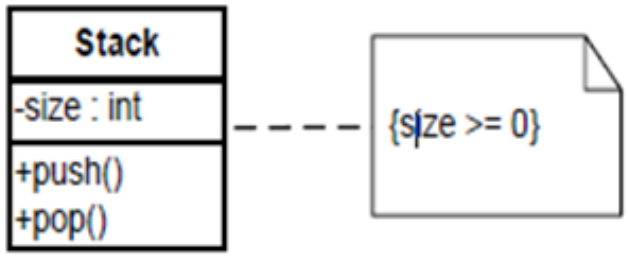


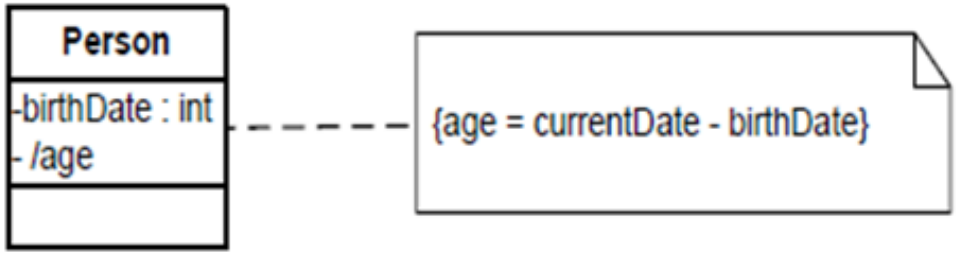
- Une expression booléenne : true ou false
- Peut être écrite :
  - Dans une note dans le diagramme UML
    - Syntaxe : {contrainte}
    - La note est liée à l'élément concerné du diagramme
    - Cela risque de rendre le diagramme illisible
  - Dans un document accompagnant le diagramme

- Trois types de contraintes :
  - **Les invariants** qui doivent être respectés en permanence
  - Des pré et post-conditions pour **une opération** :
    - **Précondition** : doit être vérifiée **avant** l'exécution de l'opération
    - **Postcondition** : doit être vérifiée **après** l'exécution de l'opération

# Contrainte OCL : exemples





# Contrainte OCL : contexte



- Une expression OCL est toujours associée à un élément du modèle (classe, opération, attribut…) : c'est le **contexte de la contrainte**.

- Il existe deux manières de spécifier le contexte d'une contrainte OCL :
  - Dans le diagramme UML : en écrivant la contrainte entre 2 accolades dans une note. L'élément pointé par la note est alors le contexte de la contrainte.
  - Dans un document accompagnant le diagramme: en utilisant le mot-clef **context**.

# OCL : contexte



- Syntaxe : **context** élément
  - Elément peut être une classe, un attribut, une opération, ….
- Exemples
  - Le contexte est la classe Compte:

    **context** Compte
  - Le contexte est l'attribut solde de la classe Compte:

    **context** Compte::solde
  - Le contexte est l'opération getSolde() de la classe Compte:

    **context** Compte::getSolde() : Integer

Pour faire référence à un élément d'une classe, il faut utiliser les **::** comme séparateur

# Contraintes : invariants

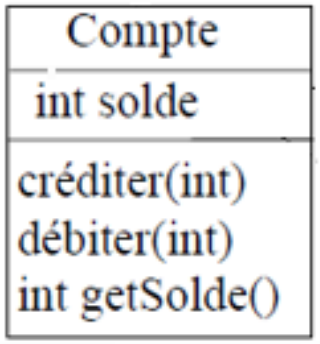


- **Invariant** : une contrainte sur un objet ou un groupe d'objets qui doit être toujours vérifiée

- Exemple :

  **context** Compte

  **inv** : solde >= 0




- L'expression OCL s'applique à la classe Compte, c'est-à-dire à toutes les instances de cette classe
- Pour toutes les instances de la classe Compte, l'attribut solde doit toujours être positif

# Contraintes : invariants



- Une expression OCL est toujours évaluée pour une instance particulière (notée **self**)
- Nommage par défaut : mot-clé **self**

  **context** Compte

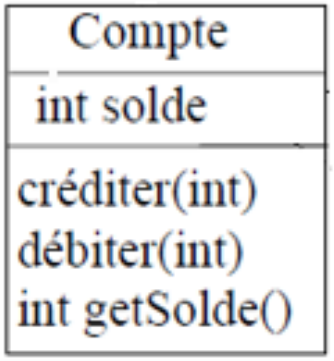
  **inv** : self.solde >= 0
- Nommage omit :

  **context** Compte

  **inv** : solde >= 0
- Nommage explicit :

  **context** p : Compte

  **inv** : p.solde >= 0



# Contraintes : pré/post conditions



- On peut spécifier les **pré/post conditions** pour les opérations
  - **pré-conditions** doivent être vraies avant l'appel de l'opération
  - **post-conditions** doivent être vraies après l'appel de l'opération
- Dans les post-conditions, on peut utiliser :
  - **result** : indiquer la valeur retournée par l'opération
  - **@pre** : indiquer la valeur d'un attribut avant l'appel de l'opération
    - **mon_attribut@pre** : référence la valeur de **mon_attribut** avant l'appel de l'opération
- Syntaxe pour préciser l'opération :

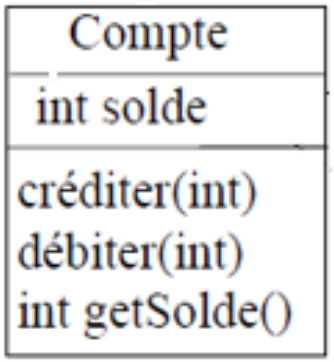
**context** ma_classe::mon_op(liste_param) : type_retour

**pre :** expressionBooléenne

**post :** expressionBooléenne

# Contraintes : exemples






- **context** Compte**::**debiter(montant: Integer)

  **pre**: montant > 0

  **post**: solde = solde**@pre** – montant

  - La somme à débiter doit être positive pour que l'appel de l'opération soit valide
  - Après l'exécution de l'opération, l'attribut solde doit avoir pour valeur la différence de sa valeur avant l'appel et de la somme passée en paramètre

- **context** Compte**::**getSolde(): Integer

  **post**: **result** = solde

- Attention : on ne décrit pas comment l'opération est réalisée mais des contraintes sur l'état avant et après son exécution

# Nommer les contraintes



- Syntaxe :

  **context** class

  **inv ConstraintName** : constraintExpression

- Exemples :

  **context** Compte

  **inv soldePositif** : self.solde > 0

  **context** Compte::debiter(montant: Integer)

  **pre montantPositif** : montant > 0

  **post montantDebite** : self.solde = self.solde@pre – montant

# Commentaires



- Syntaxe :

  **-- comment**

- Exemples :

**context** Compte

**inv** : self.solde > 0 **-- solde positif**

**context** Compte::debiter(montant: Integer)

**pre** : montant > 0 **-- montant positif**

**post** montantDebite : self.solde = self.solde@pre – montant

# Résultat d'une opération



- Une expression OCL peut être utilisée pour indiquer le résultat d'une opération de type "query" :

**context** ClasseName::operation(param1:Type1,…): retType

**body:--expression qui retourne une valeur de type retType**

- Exemple :

**context** Compte::getSolde() : Float

**body : solde** (ou self.solde)

# Valeurs initiales



- Une expression OCL peut être utilisée pour indiquer la valeur initiale d'un attribut

  **context** ClasseName::AttributeName: Type

  **init**: -- expression représentant la valeur initiale

- Exemple :

  **context** Personne::marié : Boolean

  **init**: false

# Valeurs dérivées



- Une expression OCL peut être utilisée pour indiquer la valeur dérivée d'un attribut

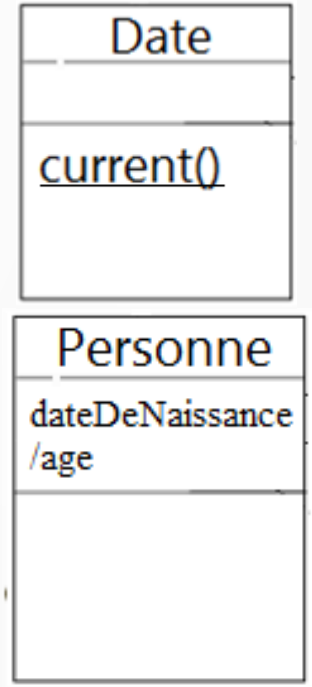
  **context** ClasseName::AttributeName: Type

  **derive**:--expression représentant la règle de dérivation

- Exemple :

  **context** Personne::age : Integer

  **derive**: Date::current() – dateDeNaissance



# Rappel



- **context** Classe
  **inv** : expressionBooléenne

- **context** Classe :: opération(paramètres) : Type
  **pre** : expressionBooléenne
  **post** : expressionBooléenne

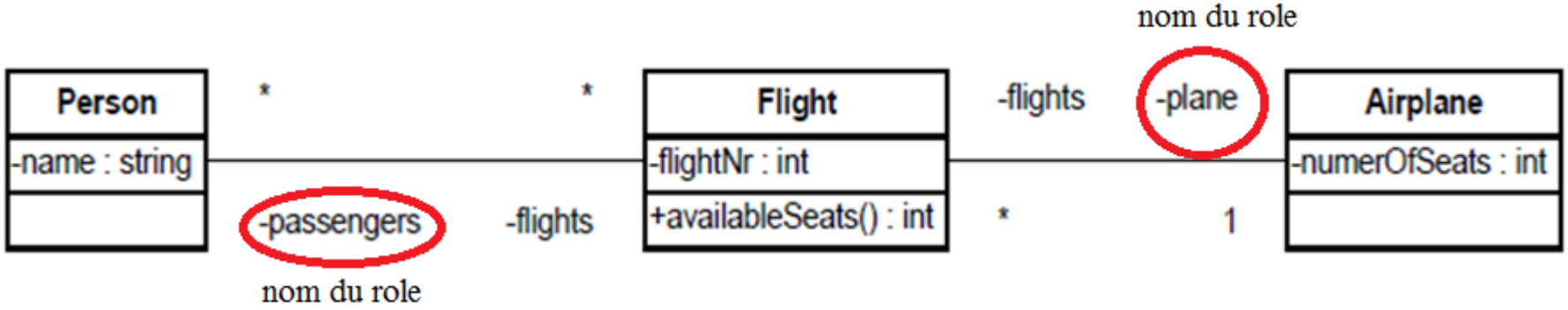
- **context** Classe :: attribut : Type
  **init** : valeur

- **context** Classe :: attribut : Type
  **derive** : expression

- **context** Classe :: opération(paramètres) : Type
  **body** : expressionOCL

# Accéder aux propriétés d'une classe





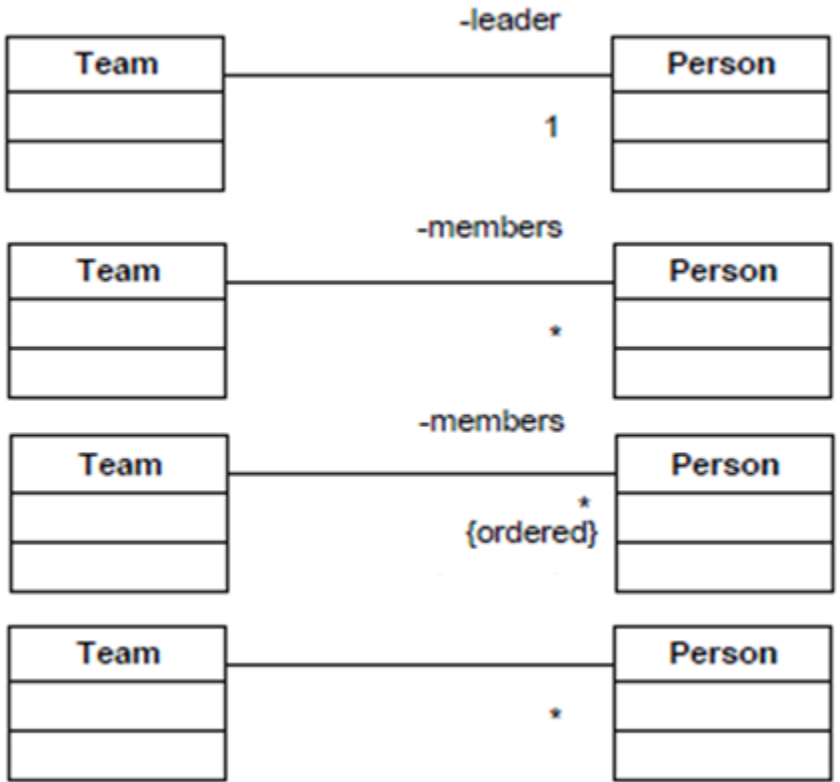

- Exemple: si Flight est le contexte (**self** est un objet Flight), pour accéder :
  - à un attribut : **self.flightNr**
  - à une opération : **self.availableSeats()**
  - aux autres objets associés à self:
    - **self.plane** : qui donne l'objet de la classe Airplane et qui est associé à l'objet self
    - **self.passengers :** qui donne les objets de la classe Person et qui sont associés à l'objet self

Noter l'importance de rôles : on utilise le rôle plutôt que le nom de l'association

# Navigabilité et Collections



- On peut naviguer à une autre classe en utilisant **le rôle de l'association** du coté de cette classe. **Si aucun rôle n'est spécifié alors on utilise le nom de la classe (en minuscule)**.
- Le résultat d'une navigation peut être un seul objet ou bien une collection d'objets




**context** Team (self est un objet Team)

- self.leader : Person (1 seul objet)
(**cardinalité = 1**)

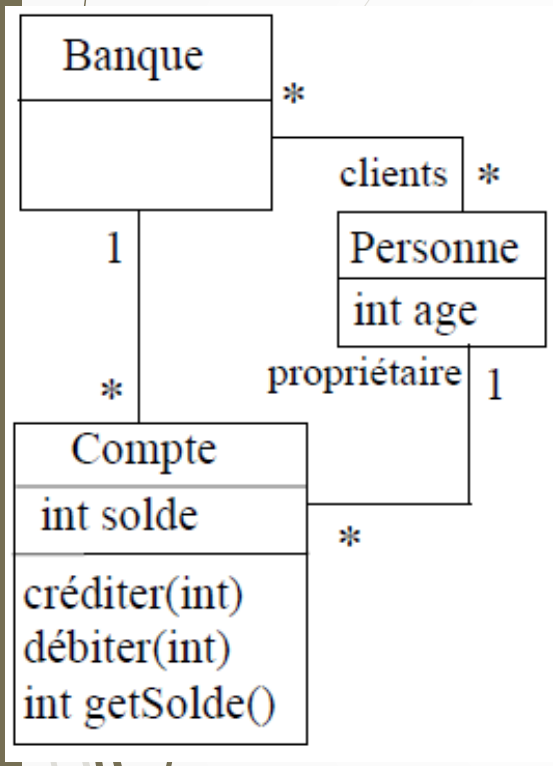
- self.members : Set(Person)
(ensemble d'objets Person)
(**cardinalité > 1**)

- self.members : OrderedSet(Person)
(ensemble ordonné d'objets Person)
**(cardinalité > 1)**

- self.person : Set(Person)

# Accès aux objets, navigation : exemple






- Contexte : la classe **Compte** (**self** est un objet Compte):
  - **self.solde** (ou **solde**): attribut référencé directement
  - **self.banque** (ou **banque**) : objet de la classe Banque (pas de rôle, on utilise le nom de la classe) associé au compte
  - **self.propriétaire** (ou **propriétaire**): objet de la classe Personne (référence via le nom de rôle d'association) associée au compte
  - **self.banque.clients** (ou **banque.clients**): ensemble des clients de la banque associée au compte
  - **self.banque.clients.age** (ou **banque.clients.age**) : ensemble des âges de tous les clients de la banque associée au compte

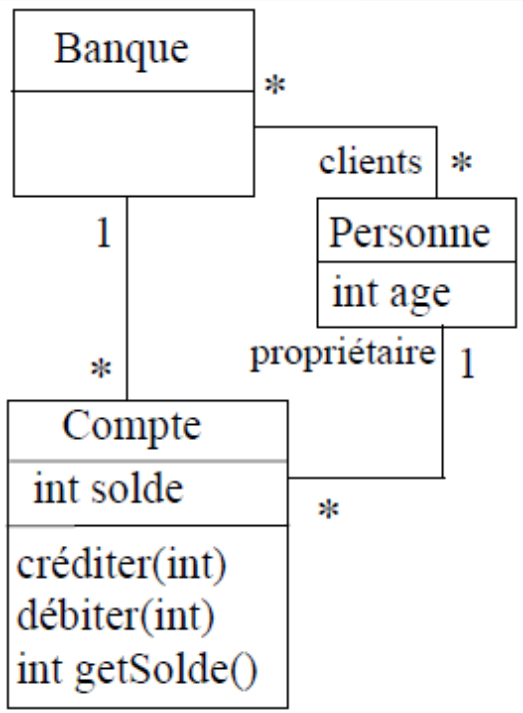
# Accès aux objets, navigation : exemple



- Le propriétaire **d'un compte** doit avoir plus de 18 ans :

  **context** Compte

  **inv**: self.propriétaire.age >= 18




- La somme des soldes **d'une personne** ne doit pas dépasser 10000 :

  **context** Personne

  **inv**: self.compte.solde **->sum()** <= 10000

# Opérations sur les éléments d'une collection



- 4 types de collection d'objets :
  - Set : ensemble au sens mathématique, pas de doublons, pas d'ordre
  - OrderedSet : un ensemble dont les éléments sont ordonnés
  - Bag : comme un Set mais avec possibilité de doublons
  - Sequence : un Bag dont les éléments sont ordonnés
- Exemples :
  - { 1, 4, 3, 5 } : Set
  - { 1, 4, 1, 3, 5, 4 } : Bag
  - { 1, 1, 3, 4, 4, 5 } : Sequence
- Possibilité de transformer un type de collection en un autre type de collection
- En OCL 2.0, possibilité de collections de collections

# Opérations sur les collections



OCL propose un ensemble de primitives utilisables sur les collections.

Syntaxe d'utilisation : **Collection -> primitive**

Une liste non exhaustive de ces primitives :

- **size()** : retourne le nombre d'éléments de la collection
- **isEmpty()** : retourne vrai si la collection est vide
- **notEmpty()** : retourne vrai si la collection n'est pas vide
- **includes(obj)** : retourne vrai si la collection inclut l'objet obj
- **excludes(obj)** : retourne vrai si la collection n'inclut pas l'objet obj
- **including(obj)** : la collection référencée doit être cette collection en incluant l'objet obj
- **excluding(obj)** : idem mais en excluant l'objet obj
- **includesAll(ens)** : retourne vrai si la collection contient tous les éléments de la collection ens
- **excludesAll(ens)** : retourne vrai si la collection ne contient aucun des éléments de la collection ens

# Opérations sur les collections : exemples

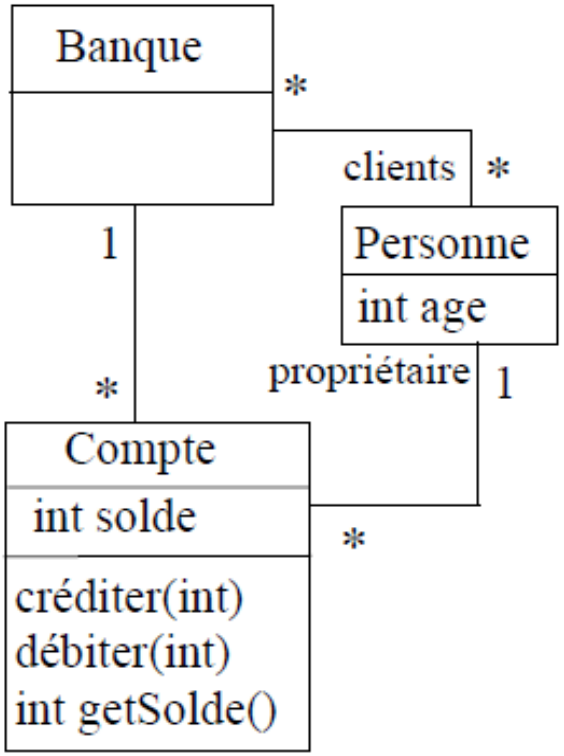


- Exemples, invariants dans le contexte de la classe **Compte**
- **Context** Compte

  **inv: self.propriétaire->notEmpty()**
- il y a au moins un objet Personne associé à un compte

- **Context** Compte

  **inv: self.propriétaire->size() = 1**
- le nombre d'objets Personne associés à un compte est de 1



# Opérations sur les collections : exemples



- **Context** Compte

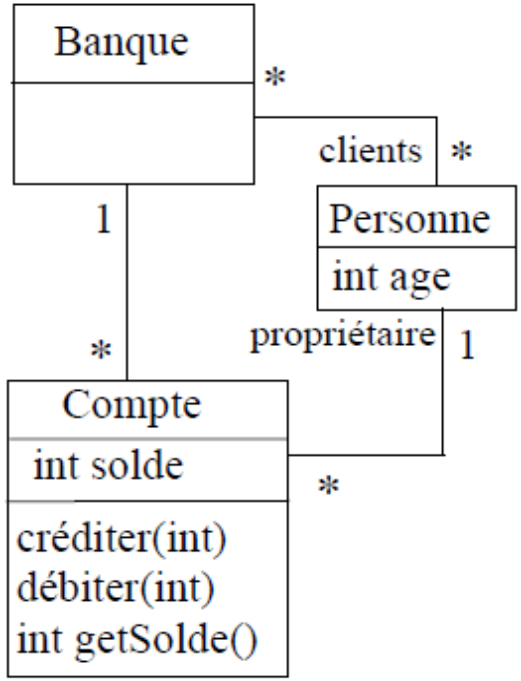
  **inv: self.banque.clients->includes(self.propriétaire)**

- l'ensemble des clients de la banque associée au compte contient le propriétaire du compte

- **Context** Compte

  **inv: self.banque.clients->size() >= 1**

- une banque associé à un compte a au moins un client



# Opérations sur les collections : exemples



- **context** Banque :: créerCompte(p : Personne) : Compte

  **post**: **result**.oclIsNew() **and** -- **result** est le résultat retourné par l'opération

  compte = compte@pre-> including(result) **and**

  p.compte = p.compte@pre-> including(result)

  - Un nouveau compte est créé. Le nouveau compte est retourné par l'opération
  - La banque doit gérer ce nouveau compte.
  - Le client passé en paramètre doit posséder ce compte.

- **oclIsNew()** : primitive indiquant qu'un objet est créé pendant l'appel de l'opération (à utiliser dans une postcondition)
- and : permet de définir plusieurs contraintes pour un invariant, une pré ou postcondition

# Opérations sur les ensembles



- **union** : retourne l'union de deux ensembles
- **intersection** : retourne l'intersection de deux ensembles

- Exemples :
  - ens1-> **intersection**(ens2)

  - contient l'intersection des éléments de ens1 et de ens2

  - (ens1-> **intersection**(ens2))-> isEmpty()

  - Les ensembles ens1 et ens2 n'ont pas d'éléments en commun

  - ens1 = ens2->**union**(ens3)

    L'ensemble ens1 doit être l'union des éléments de ens2 et de ens3

# Opérations sur les éléments d'une collection



- OCL permet de vérifier des contraintes sur chaque élément d'une collection ou de définir une sous-collection à partir d'une collection en fonction de certaines contraintes
- Primitives offrant ces services et s'appliquant sur une collection col : **col->primitive(contrainte)**
  - **select** (contrainte): retourne le sous-ensemble de la collection col dont les éléments respectent la contrainte spécifiée
  - **reject** (contrainte): idem mais ne garde que les éléments ne respectant pas la contrainte
  - **collect** (expression): retourne une collection (de taille identique) construite à partir des éléments de col. Le type des éléments contenus dans la nouvelle collection peut être différent de celui des éléments de col.
  - **exists** (contrainte): retourne vrai si au moins un élément de col respecte la contrainte spécifiée et faux sinon
  - **forAll** (contrainte): retourne vrai si tous les éléments de col respectent la contrainte spécifiée

# Opérations sur les éléments d'une collection



**Context** Banque :

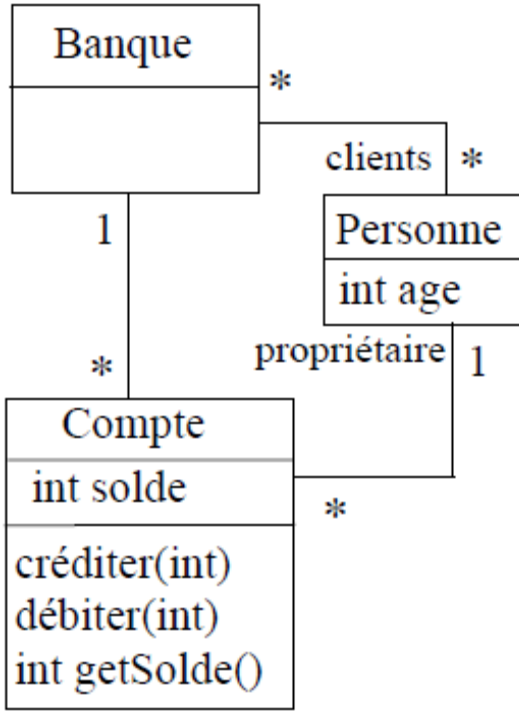
**compte->select(c : Compte| c.solde > 1000)**

OU

**compte->select( c | c.solde > 1000)**

OU

**compte->select(solde > 1000)**




- Retourne une collection contenant tous les comptes bancaires dont le solde est supérieur à 1000 €

# Opérations sur les éléments d'une collection



Collection->collect (expr) : collection

- Évalue l'expression expr pour chaque élément de la collection, et rend une autre collection, composée par les résultats de l'évaluation.
- Le résultat est un multi-ensemble (bag).

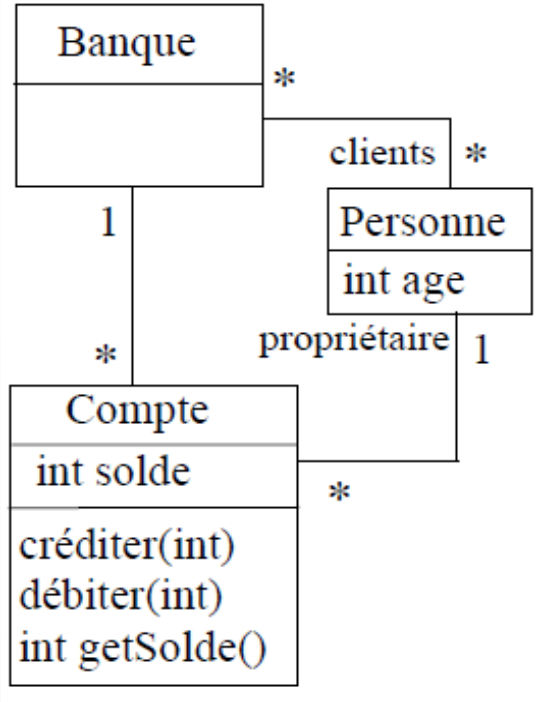
**Context** Banque :

**compte->collect( c : Compte | c.solde)**

- Retourne une collection contenant l'ensemble des soldes de tous les comptes

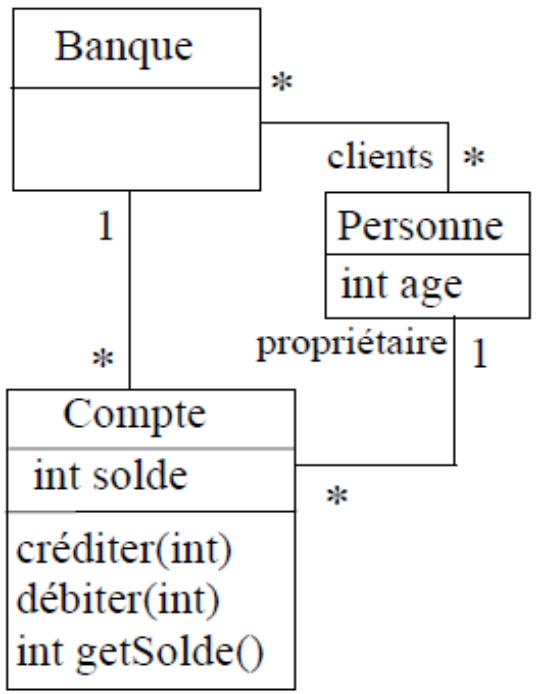
-- Raccourci:

self.compte.solde



# Opérations sur les éléments d'une collection






**compte->reject(c :Compte | c.solde > 1000)**

- Retourne une collection contenant tous les comptes bancaires dont le solde n'est pas supérieur à 1000 €

**(compte->select( solde > 1000 ))->collect( c | c.solde)**

- Retourne une collection contenant tous les soldes des comptes dont le solde est supérieur à 1000 €

# Opérations sur les éléments d'une collection

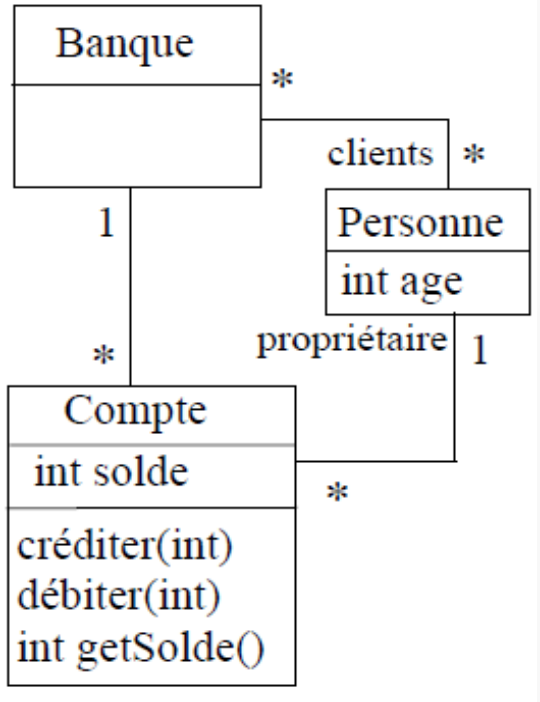


- **context** Banque

  **inv**: **not**( self.clients->exists (age < 18) )

  Ou clients->forAll (age >= 18)

  - Il n'existe pas de clients de la banque dont l'âge est inférieur à 18 ans
  - not : prend la négation d'une expression



# L'opération d'itération Iterate



- collection->**iterate(**element : Type;
  accumulateur : Type = expression-init |
  expression-avec-accumulateur-et-element**)**

- la variable *element* est l'itérateur
- l'*accumulateur* est initialisé par expression-init
- le contenu de l'accumulateur est construit par expression-avec-accumulateur-et-element pendant l'itération de tous les éléments de la collection

# L'opération d'itération Iterate



- Exemples :

  compte->collect(c | c.solde)->sum()

  -- somme des soldes; est identique à :

  compte->iterate(c;
  somme : Integer = 0 | somme + c.solde)

- collection->collect(x : T | x.propriete)

  -- est identique à :

  collection->iterate(x : T; acc : T2 = Bag{} | acc->
  including(x.propriete))

# Conditionnelles



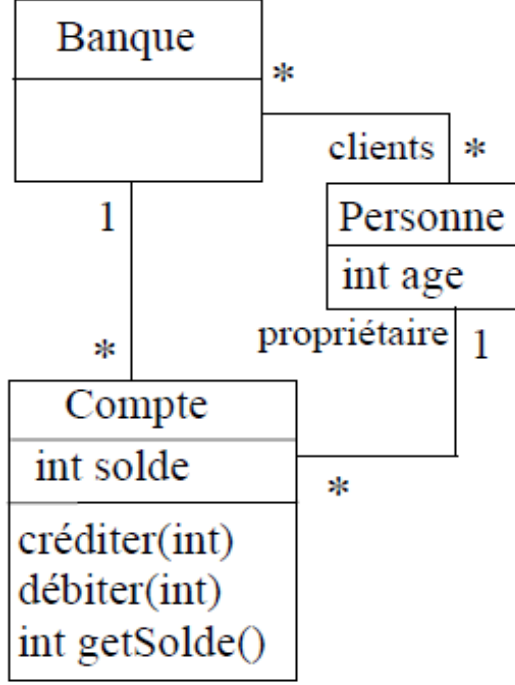
Certaines contraintes sont dépendantes d'autres contraintes. Deux formes pour gérer cela :

Syntaxe :

- **if** expr1 **then** expr2 **else** expr3 **endif** :
  - si l'expression expr1 est vraie alors expr2 doit être vraie sinon expr3 doit être vraie
- expr1 **implies** expr2 :
  - si l'expression **expr1 est vraie**, alors **expr2 doit être vraie** également. Si expr1 est fausse, alors l'expression complète est vraie
  - Equivalent à **if** expr1 **then** expr2 **endif**

# Conditionnelles






Exemples :

**context** Personne

**inv** : **if** self.age < 18 **then** self.compte->isEmpty()

**else** self.compte->notEmpty()

**endif**

- Une **personne** de moins de 18 ans n'a pas de compte bancaire alors qu'une personne de plus de 18 ans possède au moins un compte

**context** Personne

**inv**:self.compte->notEmpty() **implies** self.banque->notEmpty()

- Si une **personne** possède au moins un compte bancaire, alors elle est cliente d'au moins une banque

# Variables



- Les variables peuvent être utilisées pour améliorer la compréhension des contraintes complexes
- Syntaxe : **let** var = expression **in** OCLexpression

  **context** Personne

  **inv**: **let** argent = self.compte.solde->sum() **in**

  self.age >= 18 **implies** argent > 0

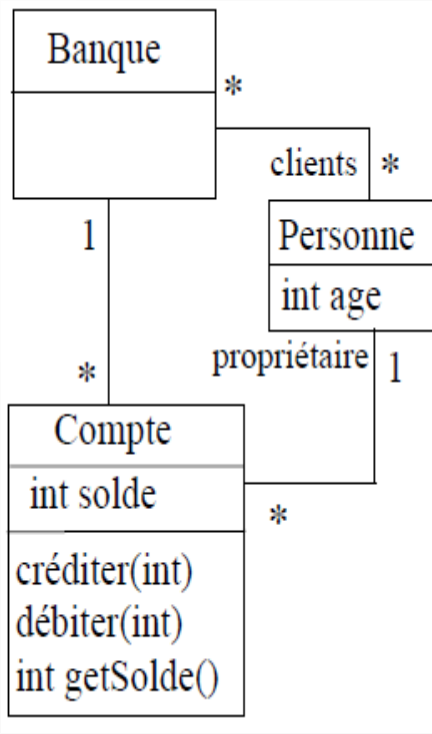
  - Une personne majeure doit avoir de l'argent

- Pour les rendre accessible partout : **def**

  **context** Personne

  **def**: argent : Integer = self.compte.solde->sum()

  **inv**: self.age >= 18 **implies** argent > 0



# Opérations sur les éléments d'une collection



- **allInstances()** : primitive s'appliquant sur une classe (et non pas un objet) et retournant toutes les instances de la classe référencée (ici la classe Personne)

- **context** Personne

  **inv**: Personne.**allInstances**()->forAll(p1, p2 |

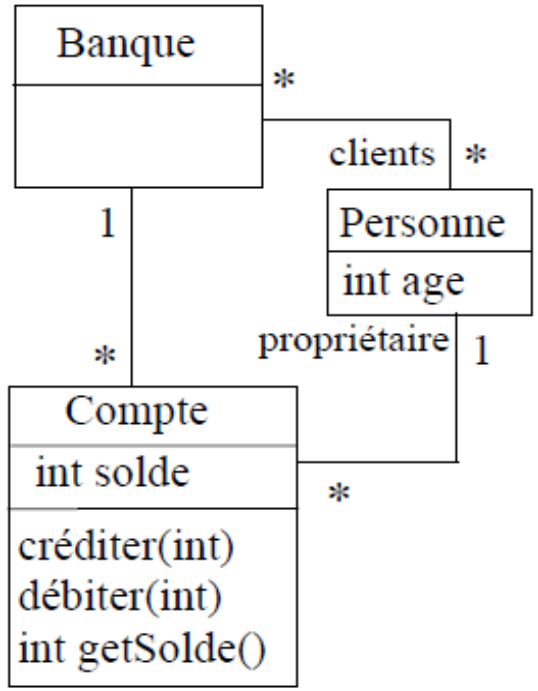
  p1 <> p2 **implies** p1.nom <> p2.nom)

  - Il n'existe pas deux instances de la classe Personne pour lesquelles l'attribut nom a la même valeur : deux personnes différentes ont des noms différents

# Appels d'opération des classes



- Dans une contrainte OCL, il est possible d'accéder à un attribut ou d'utiliser une opération d'une classe.




- Exemple :

  **context** Banque

  **inv**: compte->forAll( c | c.getSolde() > 0)

  - Les soldes de tous les comptes doivent être > 0.
  - getSolde() est une opération de la classe Compte.

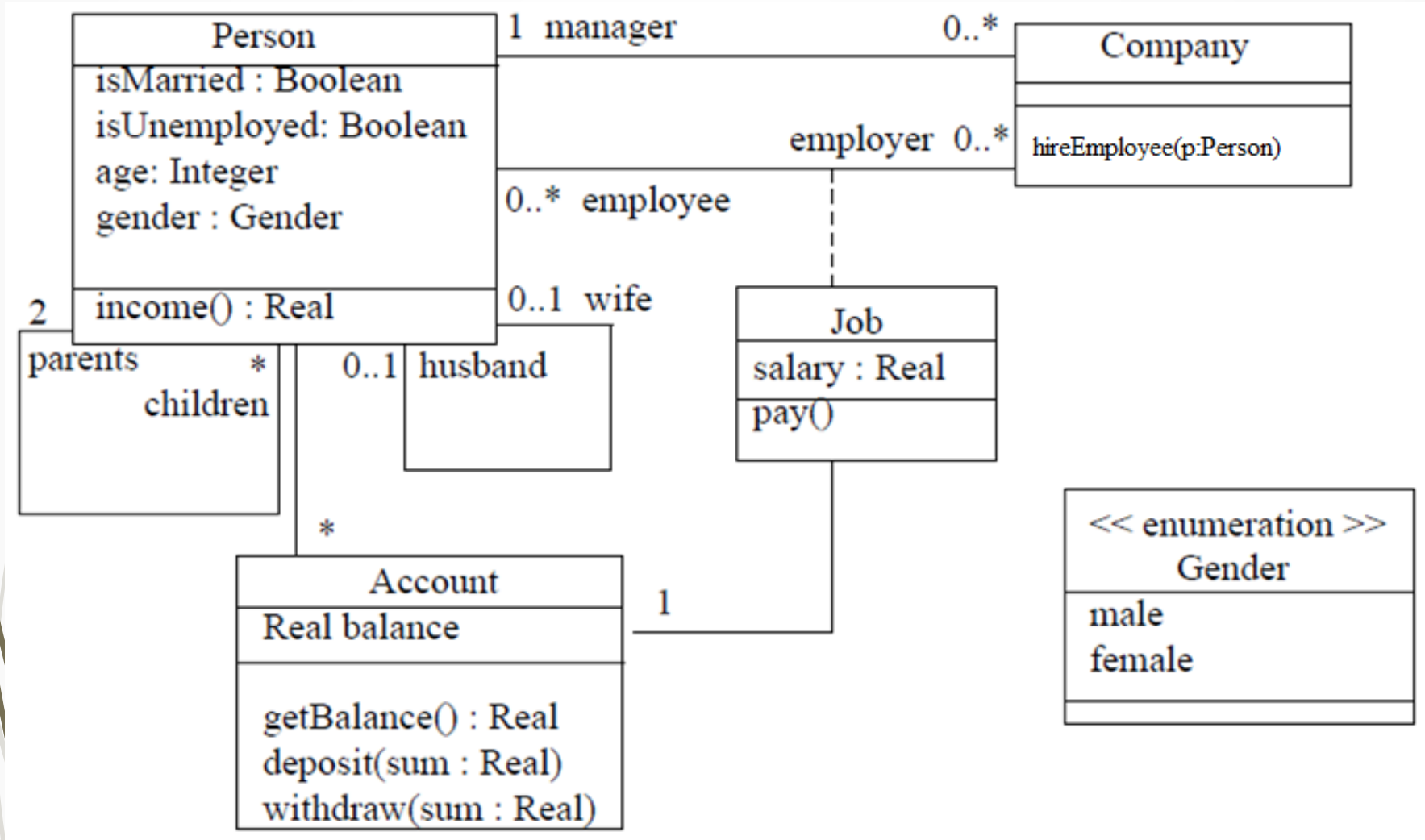
# C'est quoi OCL?



- Pourquoi OCL ? Introduction par l'exemple
- Les principaux concepts d'OCL
- **Exemple d'application**
- **Complément OCL**

# Diagramme de classe

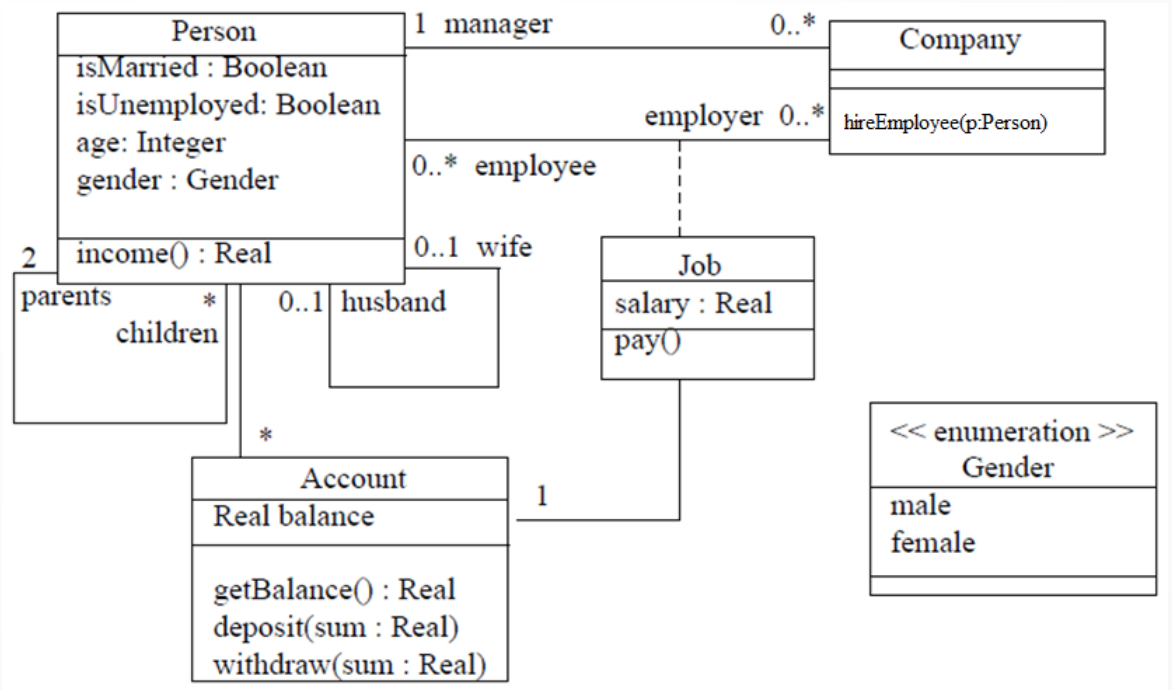




# Contraintes sur employés d'une compagnie



- Dans une **compagnie**, un manager doit travailler et avoir plus de 40 ans. Le nombre d'employés d'une compagnie est non nul.
- Un employé qui est embauché n'appartenait pas déjà à la **compagnie**



# Contraintes sur employés d'une compagnie



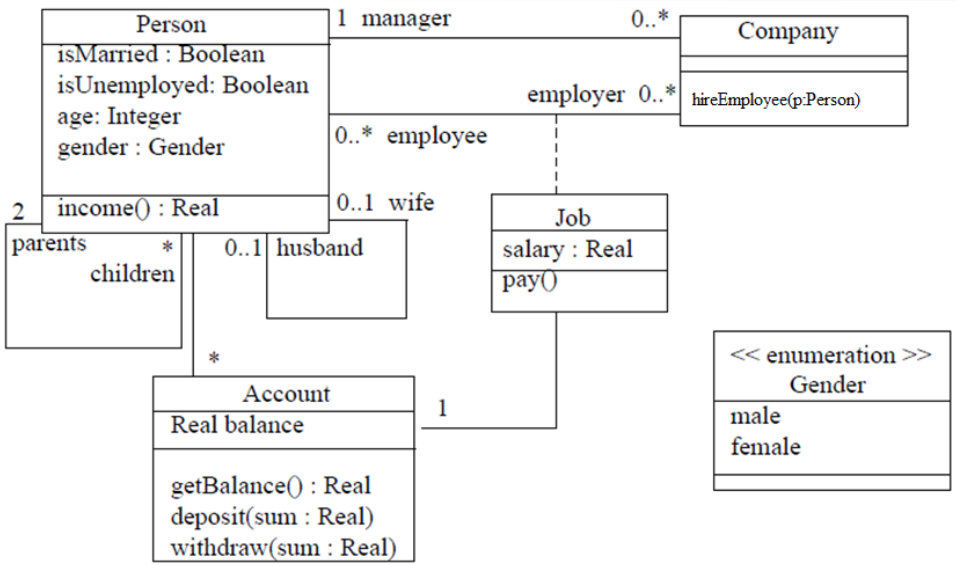
- Dans une **compagnie**, un manager doit travailler et avoir plus de 40 ans. Le nombre d'employés d'une compagnie est non nul.

**context** Company:

**inv**: self.manager.isUnemployed = false **and**

self.manager.age>40 **and**

self.employee-> notEmpty()



# Contraintes de mariage

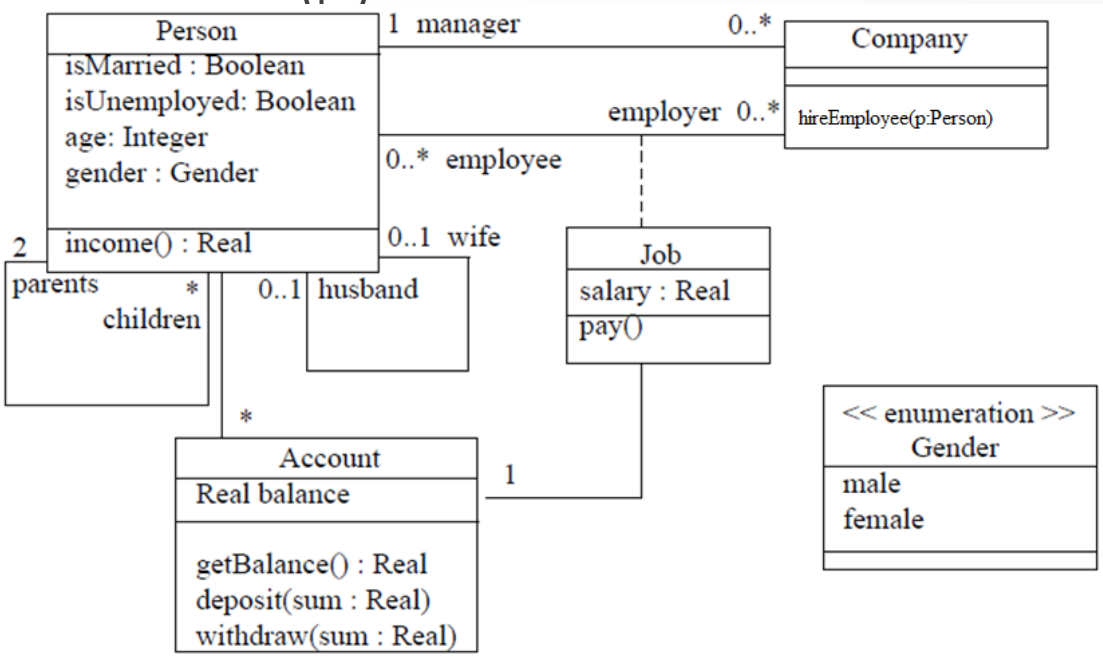


- Un employé qui est embauché n'appartenait pas déjà à la **compagnie**

**context** Company::hireEmployee(p : Person)

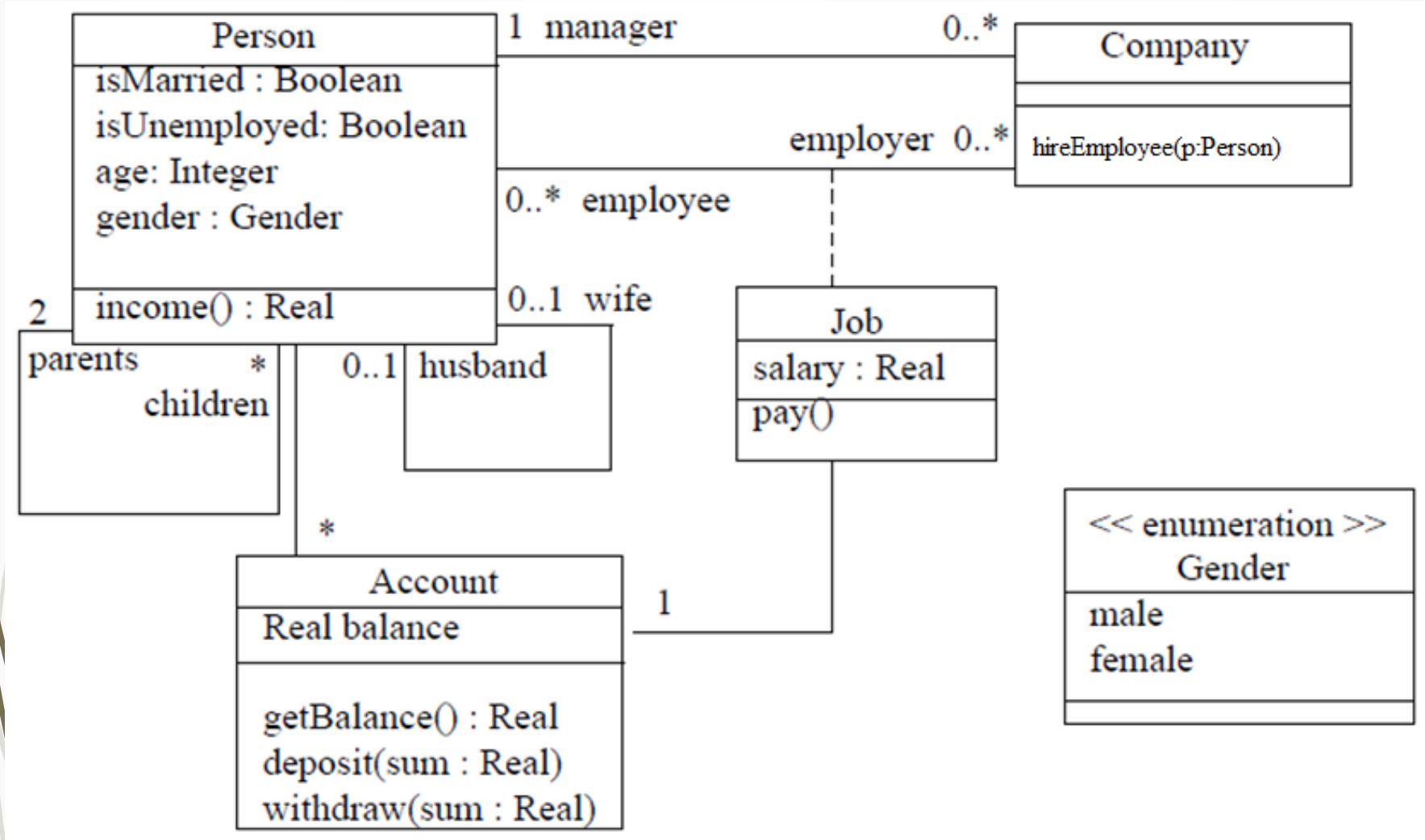
**pre**: self.employee->excludes(p)

**post**: self.employee->includes(p)



# Diagramme de classe





# Revenus selon l'âge



- Selon l'âge de la personne, ses revenus (l'opération income() ) sont :
  - 1% des revenus des parents quand elle est mineure (argent de poche)
  - Ses salaires quand elle est majeure

**context** Person::income() : Real

**post**:

if self.age < 18 then

    **result** = (self.parents.job.salary->sum()) * 1%

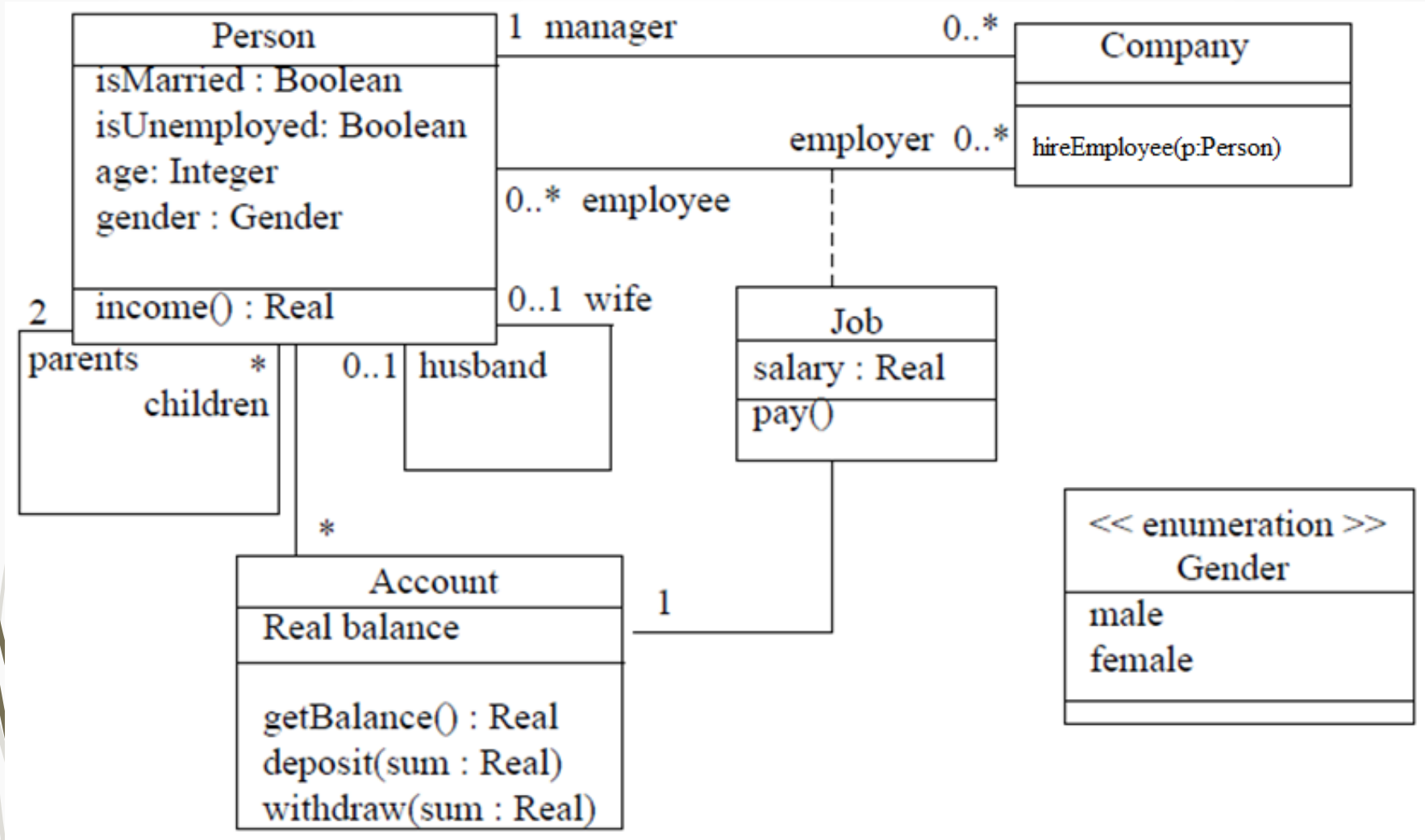
else

    **result** = self.job.salary->sum()

endif

# Diagramme de classe





# Contraintes sur les parents/enfants



- Tous les enfants d'une personne ont bien cette personne comme parent.
- **context** Person

  **inv**: --self : parent

  self.children->notEmpty() **implies** self.children -> forAll ( p : Person | p.parents -> includes(self))

- Tous les parents d'une personne ont bien cette personne comme enfant.
- **context** Person

  **inv**: --self : enfant

  self.parents -> forAll ( p : Person |p.children -> includes (self))

# Lien salaire/chomage pour une personne



- Un enfant a un père et une mère

**context** Person
**def**: parent1 = parents ->at(1)
**def**: parent2 = parents ->at(2)

**context** Person
**inv**:
if parent1.gender = Gender::Male (ou #Male)
then --parent1 est un homme
parent2.gender = Gender::Female
else --parent1est une femme
parent2.gender = Gender::Male
endif