Université Abdelhamid Ibn Badis Mostaganem
Faculté des sciences exactes et informatique

Ingénierie Dirigée par les Modèles (IDM)

OCL

Introduction

- IDM s'intéresse à la création de modèles productifs
  - Cela nécessite la vérification de ces modèles dans la phase de spécification et conception de modèle avant de passer au développement (code source)
- Deux moyens de vérifications de modèles
  - Le standard OCL (programmes simples non critiques)
  - Model checking (systèmes interactifs, critiques)

Pourquoi OCL?

- Diagrammes UML **insuffisants** pour spécifier complètement une application
  - Nécessité de rajouter des **contraintes**
- **Comment exprimer ces contraintes ?**
  - Langue **naturelle** mais manque de précision, compréhension pouvant être **ambiguë**
  - OCL est un langage proposé pour limiter les ambiguïtés du langage naturel, tout en restant accessible.
- La syntaxe d'OCL est utilisé pour exprimer d'autres langages dans IDM (transformation)

OCL?

- OCL (**Object Constraint Language**) est un langage **formel** d'expression de contraintes qu'UML utilise pour formaliser l'expression des contraintes.
- C'est un standard de l'OMG (IDM) qui peut s'appliquer sur tout type de modèle, indépendant d'un langage de modélisation donné
- Il est bien adapté aux diagrammes d'UML, et en particulier au diagramme de classes.
- OCL permet de spécifier des contraintes sur l'état d'un objet ou d'un ensemble d'objets.

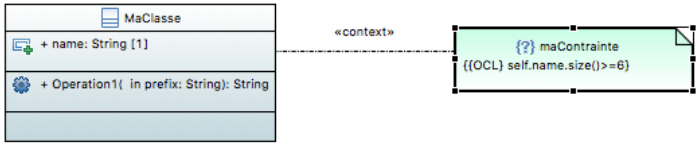
OCL?

- Les contraintes OCL se font à travers :
  - des **invariants** sur des classes
  - des **préconditions** (doivent être vérifiées avant l'exécution) à l'exécution d'opérations
  - des **postconditions** (doivent être vérifiées après l'exécution) à l'exécution d'opérations
  - des gardes sur des transitions de diagrammes d'états transitions ou des messages de diagrammes d'interaction
  - des ensembles d'objets destinataires pour un envoi de message
  - des attributs dérivés
  - des stéréotypes, etc.

Syntaxe des contraintes OCL

Contexte (context)

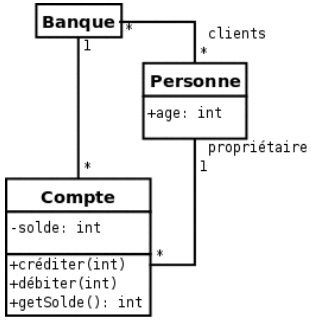
- Une contrainte est toujours associée à un **élément de modèle**. C'est cet élément qui constitue le **contexte** de la contrainte.
- Il existe deux manières pour spécifier le contexte d'une contrainte OCL :
  - Representation graphique dans les diagrammes UML: en écrivant la contrainte entre accolades ({ }) dans une **note**.



  - Dans un document accompagnant le diagramme en utilisant le mot-clef **context**

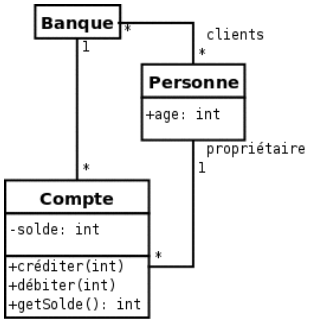
Contexte (context)

- **Syntaxe:** **context** <élément>
  - <élément> peut être une classe, une opération, etc.
  - Pour faire référence à un **élément op** d'un **classeur C** il faut utiliser les **: :** comme séparateur (comme **C : :op**).
- **Exemples:**
  - **context** Compte
    - Le contexte est la classe Compte
  - **context** Compte **: :** getSolde()
    - Le contexte est l'opération getSolde() de la classe Compte



Invariants (inv)

- Un **invariant** exprime une **contrainte prédicative** sur un objet, ou un groupe d'objets, qui doit être respectée en permanence.
- **Syntaxe** : **inv :** <expression_logique>
  - <expression_logique> doit toujours être vraie.
- **Exemple** :
  - **context** Compte
    inv : solde > 0
  - Le solde d'un compte doit être toujours positif



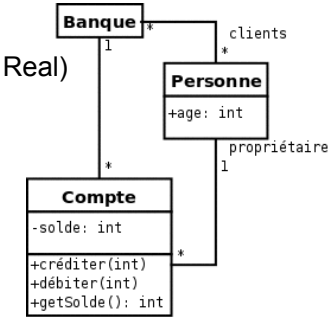
Préconditions (pre)

- Une **précondition** permet de spécifier une **contrainte prédicative** qui doit être vérifiée **avant** l'appel d'une **opération**.
- **Syntaxe:**
  - **pre :** <expression_logique>
  - <expression_logique> est une expression qui doit être toujours vraie.

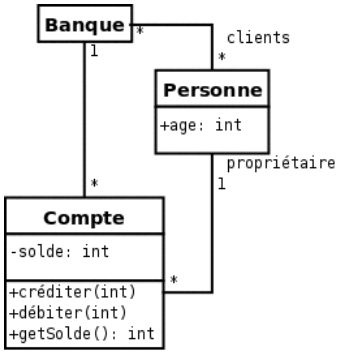
Postconditions (post)

- Une **postcondition** permet de spécifier une **contrainte prédicative** qui doit être vérifiée **après** l'appel d'une **opération**.
- **Syntaxe**
  - **post :** <expression_logique>
  - <expression_logique> est une expression qui doit être toujours vraie.
- Deux éléments particuliers sont utilisables:
  - l'attribut **result** qui désigne la valeur retournée par l'opération
  - et **<nom_attribut>@pre** qui désigne la valeur de l'attribut <nom_attribut> avant l'appel de l'opération.

- **Exemple 1:**
  - Pour la **méthode débiter** de la classe Compte, la somme à débiter doit être positive pour que l'appel de l'opération soit valide
  - et 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 **::**débiter(somme : Real)
    pre : somme > 0
    post : solde = **solde@pre** - somme



- **Exemple 2:**
  - Le résultat de l'appel de l'opération getSolde doit être égal à l'attribut solde.
  - **context** Compte **: :**getSolde() : Real
    post : result = solde



Résultat d'une méthode (body)

- Ce type de contrainte permet de définir directement le **résultat** d'une opération.
- **Syntaxe : body :** <requête>
  - <requête> le résultat de requête doit être de même type que le résultat de l'opération désignée par le contexte.
- **Exemple:** Le résultat de l'opération getSolde doit être égal à l'attribut solde.
  - Premiere solution avec post :
    - **context** Compte : :getSolde() : Real
      post : result = solde
  - Une autre solution :
    - **context** Compte : :getSolde() : Real
      body : solde

Types et opérations utilisables dans les expressions OCL

Types et opérations utilisables dans les expressions OCL

- Types et opérateurs prédéfinis
- Types du modèle UML
- Collections

Types et opérateurs prédéfinis

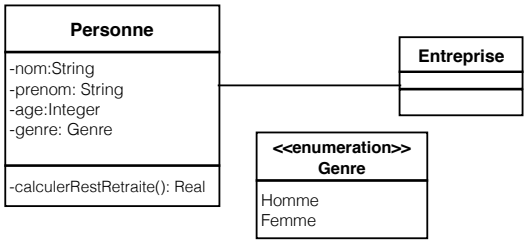
- OCL possède des types prédéfinis et des opérations prédéfinies sur ces types. qui sont indépendants du modèle
- Les **types** dans OCL sont :
  - Boolean
  - Integer
  - Real
  - String
- Les **opérateurs** les plus communs sont :
  - and ; or ; xor ; not ;
  - implies ; if-then-else-endif ;
  - + ; − ; / ; abs() ;
  - concat() ; size() ; substring() ;
  - ...

Types du modèle UML

- Toute expression OCL est écrite dans le contexte d'un modèle UML donné.
- Tous les classeurs de ce modèle sont des **types** dans les expressions OCL attachées à ce modèle.
- Pour le **type énuméré**, une contrainte OCL peut référencer une valeur de ce type de la manière suivante :
  - <nom_type_enuméré> **: :** valeur

Types du modèle UML

- Par exemple, la classe Personne possède un attribut genre de type Genre. On peut donc écrire la contrainte :



- **context** Personne
  inv : genre = Genre **: :femme**
  - Dans ce cas, toutes les personnes doivent être des femmes.

Collections

- OCL définit également la notion d'ensemble sous le terme générique de **collection**
- Il existe plusieurs sous-types du type abstrait **Collection** :
  - **Set** (Ensemble): collection non ordonnée d'éléments uniques (c.a.d pas d'élément en double)
  - **OrderedSet** (Ensemble ordonné): est un Ensemble ordonné
  - **Bag** (Sac): collection non ordonnée d'éléments identifiables (c.a.d comme un ensemble, mais pouvant comporter des doublons)
  - **Sequence** (Séquence): collection ordonnée d'éléments identifiables.

Accès aux caractéristiques et aux objets

Accès aux caractéristiques et aux objets

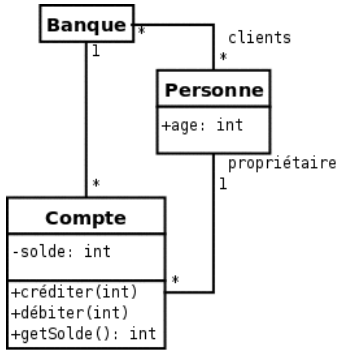
- Dans une contrainte OCL associée à un objet, on peut :
  - Accéder à l'état interne de cet objet (ses propriétés)
  - Naviguer dans le diagramme : accéder de manière **transitive** à tous les objets (et leur état) avec qui il est en relation
- Pour accéder au **attributs** ou **paramètres d'une opération** de l'objet on utilise leur **nom** directement (et/ou avec Self)

Self

- Pour faire référence à un **attribut** ou une **opération** de l'objet désigné par le contexte, il suffit soit d'utiliser:
  1. Le **nom** de cet élément.
     - Quand c'est une **opération** avec des paramètres, il faut les préciser aussi entre les parenthèses.
  2. L'expression **self** qui désigne l'objet: **self.<propriété>.**

Self

- Par exemple, dans le contexte de la classe **Compte**, on peut utiliser les expressions suivantes :
  - solde ;
  - self.solde ;
  - getSolde() ;
  - self.getSolde() ;
  - débiter(1000) ;
  - self.débiter(1000).

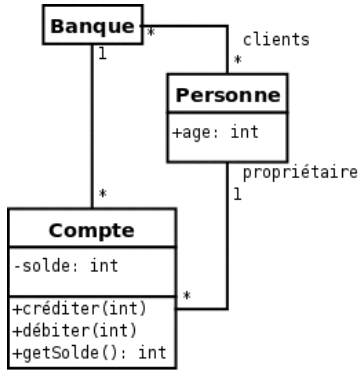


Accès aux caractéristiques et aux objets

- Dans une contrainte OCL associée à un objet, on peut :
  - Accéder à l'état interne de cet objet (ses propriétés)
  - **Naviguer dans le diagramme : accéder de manière transitive à tous les objets (et leur état) avec qui il est en relation**
- Pour accéder au **Objet(s) en association** on utilise au choix :
  - Le **nom de la classe associée** (avec la première lettre en minuscule)
  - Le **nom de l'association** si elle nommée
  - Le **nom du rôle** d'association du côté de la classe vers laquelle on navigue

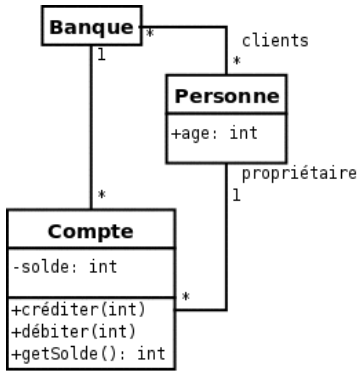
Exemples de Navigation

- Dans contexte de la classe **Compte** (context Compte):
  - **solde** : attribut référencé directement
  - **banque** : objet de la classe Banque (référence via le nom de la classe) associé au compte
  - **propriétaire** : objet de la classe Personne (référence via le nom de rôle d'association) associée au compte



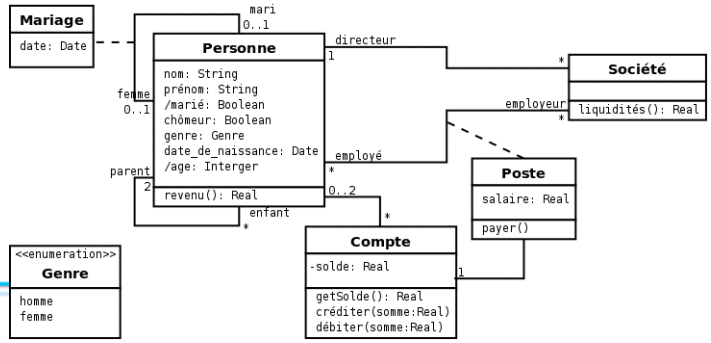
Exemples de Navigation

- Dans contexte de la classe Compte (context Compte):
  - **banque.clients** : ensemble des clients de la banque associée au compte (référence par transitivité)
  - **banque.clients.age** : ensemble des âges de tous les clients de la banque associée au compte
  - **self.propriétaire.age >= 18** Le propriétaire d'un compte doit avoir plus de 18 ans



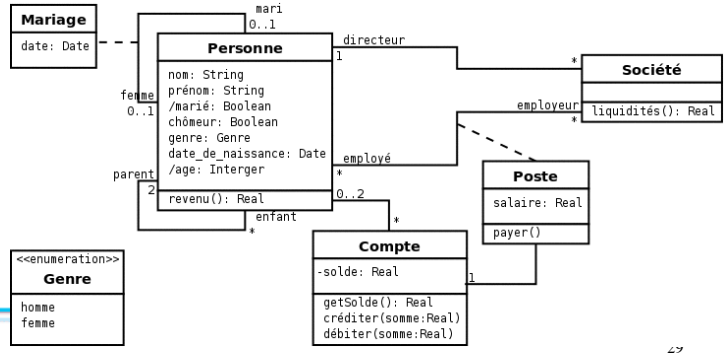
Navigation vers une classe association

- Pour naviguer vers une classe association, il faut utiliser la notation pointée classique en précisant le nom de la classe association en minuscules.
- Par exemple, dans le **contexte** de la classe **Société** (context Société), pour accéder au salaire de tous les employés, il faut écrire :
  - **self.poste.salaire**



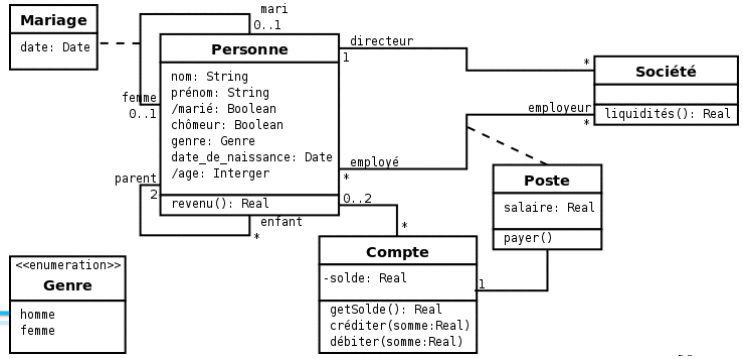
Navigation vers une classe association

- Cependant, dans le cas où l'association est **réflexive** (exp. Mariage), il faut en plus préciser par quelle extrémité il faut emprunter l'association.
- Pour cela, on précise le nom de rôle de l'une des extrémités de l'association entre crochets ([ ]) derrière le nom de la classe association.



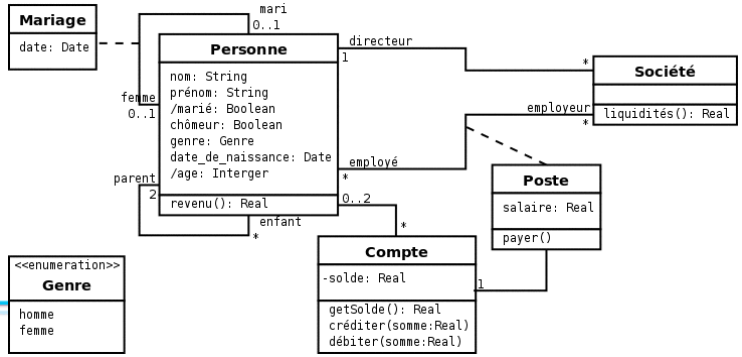
Navigation vers une classe association

- Par exemple, dans le **contexte** de la classe **Personne** (context Personne), pour accéder à la date de mariage de toutes les femmes, il faut écrire :
  - **self.mariage[femme].date**



Navigation depuis une classe association

- Par définition, naviguer depuis une classe association vers une classe participante produit toujours comme résultat un objet unique.
- Exemple :
  context Poste
  inv : self.employé.age > 21



Opérations sur objets et collections

Opérations sur objets et collections

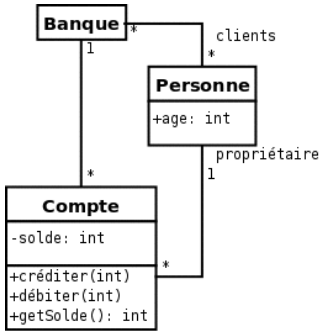
- OCL propose un ensemble de primitives utilisables sur les collections.
- Syntaxe d'utilisation : **objetOuCollection -> primitive**
- Voici quelques primitives :
  - **size()** : retourne le nombre d'éléments de la collection
  - **isEmpty()** : retourne vrai si la collection est vide
  - **notEmpty()** : retourne vrai si la collection n'est pas vide
  - **count(obj)** : le nombre d'occurrences de l'objet obj dans la collection
  - **sum()** retourne la somme des éléments

Opérations sur objets et collections

- **includes(obj)** : vrai si la collection inclut l'objet obj
- **excludes(obj)** : 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(col)** : la collection contient tous les éléments de la collection col
- **excludesAll(col)** : la collection ne contient aucun des éléments de la collection col

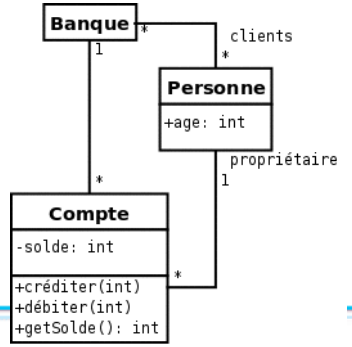
Exemples

- Les invariants dans le **contexte** de la classe **Compte**
- **propriétaire -> size() = 1** : le nombre d'objets Personne associés à un compte est de 1.
  - Vrai par principe à cause de la cardinalité de 1 qui doit être respectée.
- **banque.clients -> size() >= 1** : une banque a au moins un client



Exemples

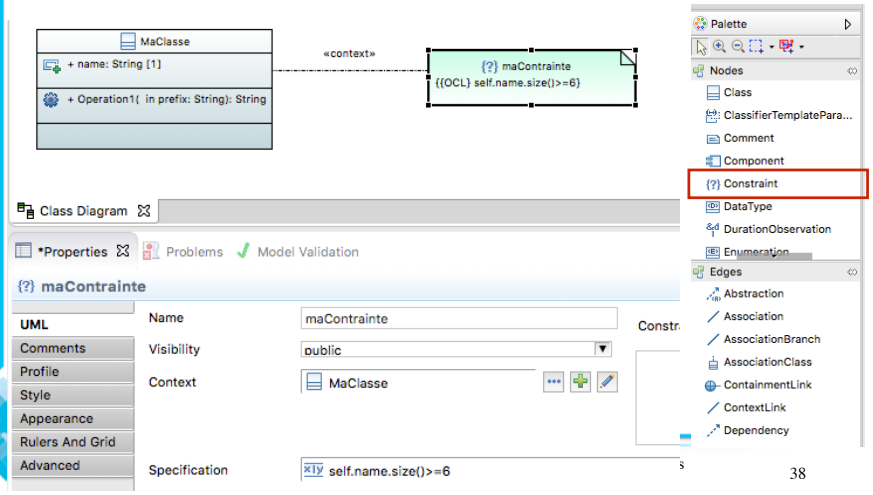
- Les invariants dans le **contexte** de la classe **Compte**
- **banque.clients -> includes(self.propriétaire)** : l'ensemble des clients de la banque associée au compte contient le propriétaire du compte
- **banque.clients.compte -> includes(self)** : le compte appartient à un des clients de sa banque



Contraintes OCL appliquées sur un modèle avec Papyrus

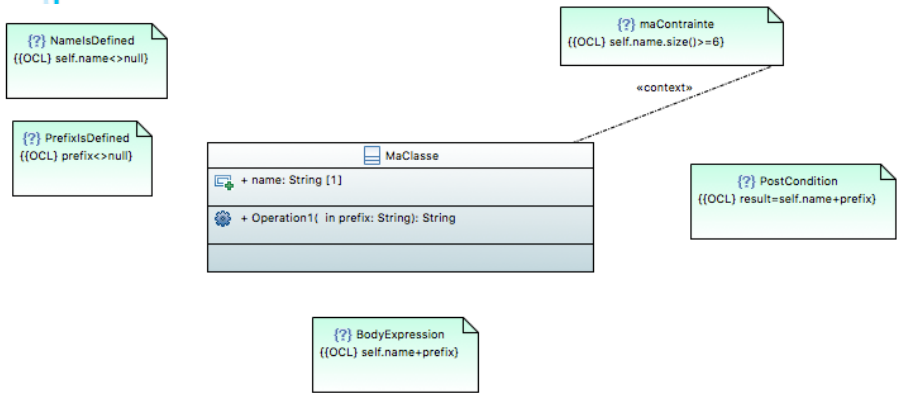
Invariant sur une classe

- Créer un noeud **Constraint** et associer le à une classe qui sera son contexte. Vous pouvez donner un nom à votre contrainte. Dans la vue Propriétés: Définissez la spécification de la contrainte puis clique droit sur la contrainte: OCL -> Validate

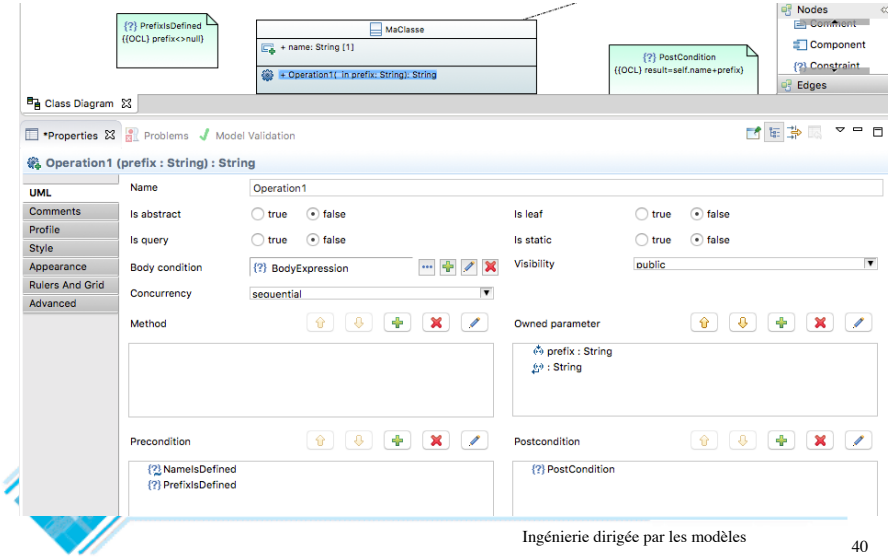


Invariant sur une opération:
Body, Pre et Post condition

- Créer les noeuds **Constraint** et faire ESC. Clique droit sur l'opération Opération1 puis rajouter les types de contraintes



Invariant sur une opération:
Body, Pre et Post condition



Contraintes OCL appliquées sur un profil UML avec Papyrus

Contraintes sur un profil

- Une contrainte dans un profil est similaire à une contrainte d'invariant sur une classe.
- Cependant, puis que le profil est dans le niveau M2, il peut être évalué au niveau M1 pour vérifier si un diagramme de classe est consistant.
- Exemple:
  - Stereotype comme extension de la méta-classe NamedElement
  - Ce stéréotype contient un méta-attribut: MaxCount qui ne soit pas dépasser la valeur 10

Exemple