Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Centre Universitaire de Mila
Laboratoire MISC, Université Constantine 2–Abdelhamid Mehri

Institut des Sciences et de la Technologie

Département: MI, Master 1 : STIC

Module: Ingénierie Des Logiciels (IDL)

Chapitre 05: Introduction à OCL

Dr MEGHZILI Said
Département MI
s.meghzili@univ-mila.dz

Sommaire

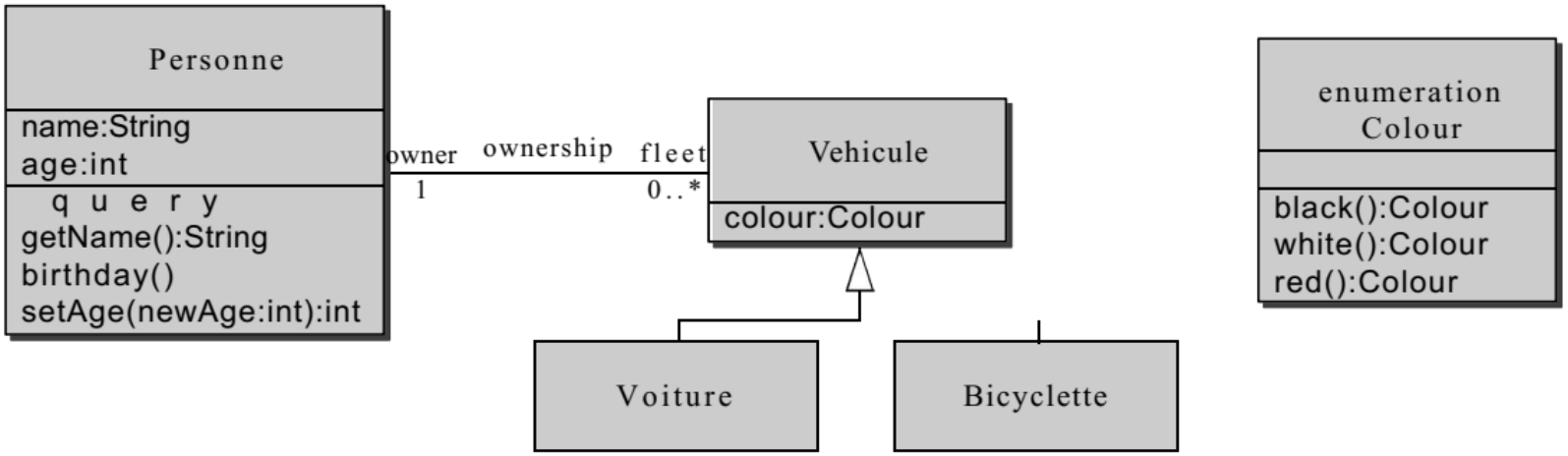
1. **Introduction**
2. **Typologie des contraintes**
3. **Types de base et opérations**
4. **Accès aux objets et navigation**
5. **Opérations pour les collections OCL**
6. ***Conclusion***

1. Introduction

- OCL est un langage formel pour écrire des expressions de **manière précise**
- Il est basé sur la logique des prédicats du premier ordre
- Les contraintes OCL ont une sémantique formelle, par conséquent, peuvent être utilisées pour réduire l'ambiguïté dans les modèles UML
  - **Exemple:** L'age d'une personne doit être supérieur ou égal à 18 ans.
- Prend en charge les concepts d'objets.
- Mais - OCL n'est pas un langage de programmation:
  - Aucun flux de contrôle.
  - Pas d'effets secondaires.
- Pourquoi OCL? Parce qu'UML ne suffit pas!

1. Introduction

- UML ne suffit pas!

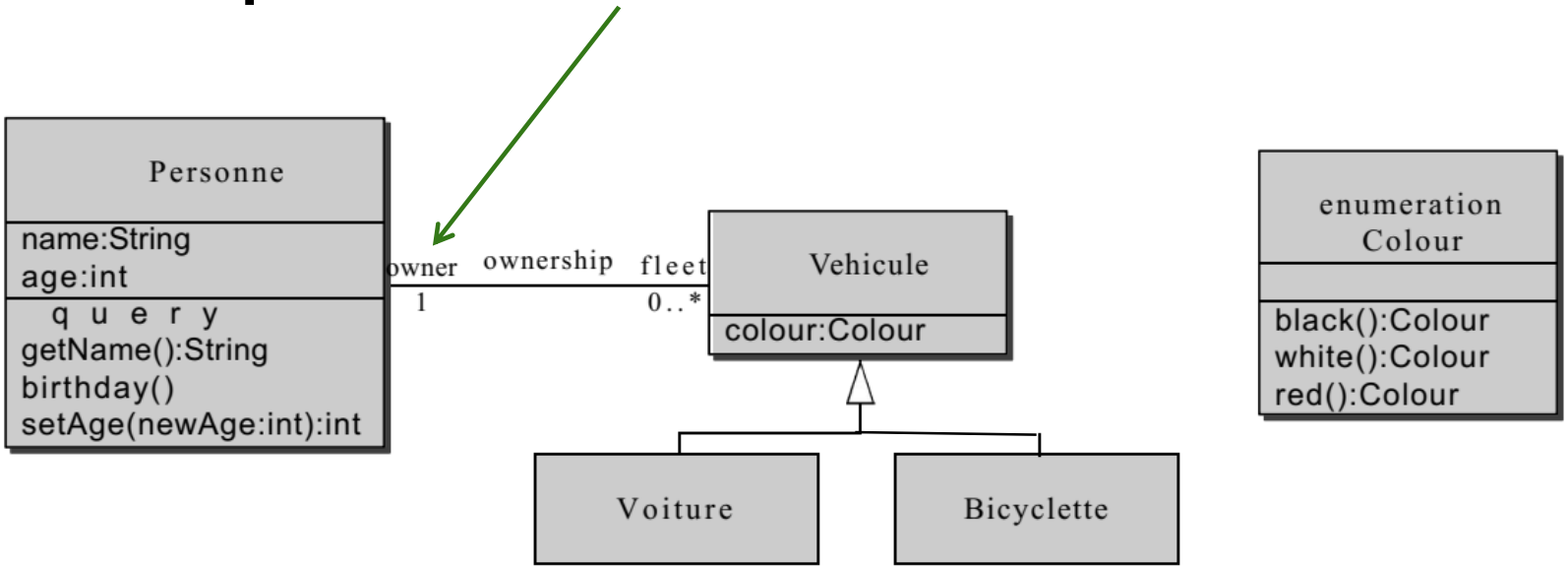


Contraintes !

- **Nombre possible de propriétaires qu'une voiture peut avoir**
- **Âge requis des propriétaires de voitures**
- **Exigence qu'une personne puisse posséder au plus une voiture noire**

1. Introduction

- **Exemples d'OCL**

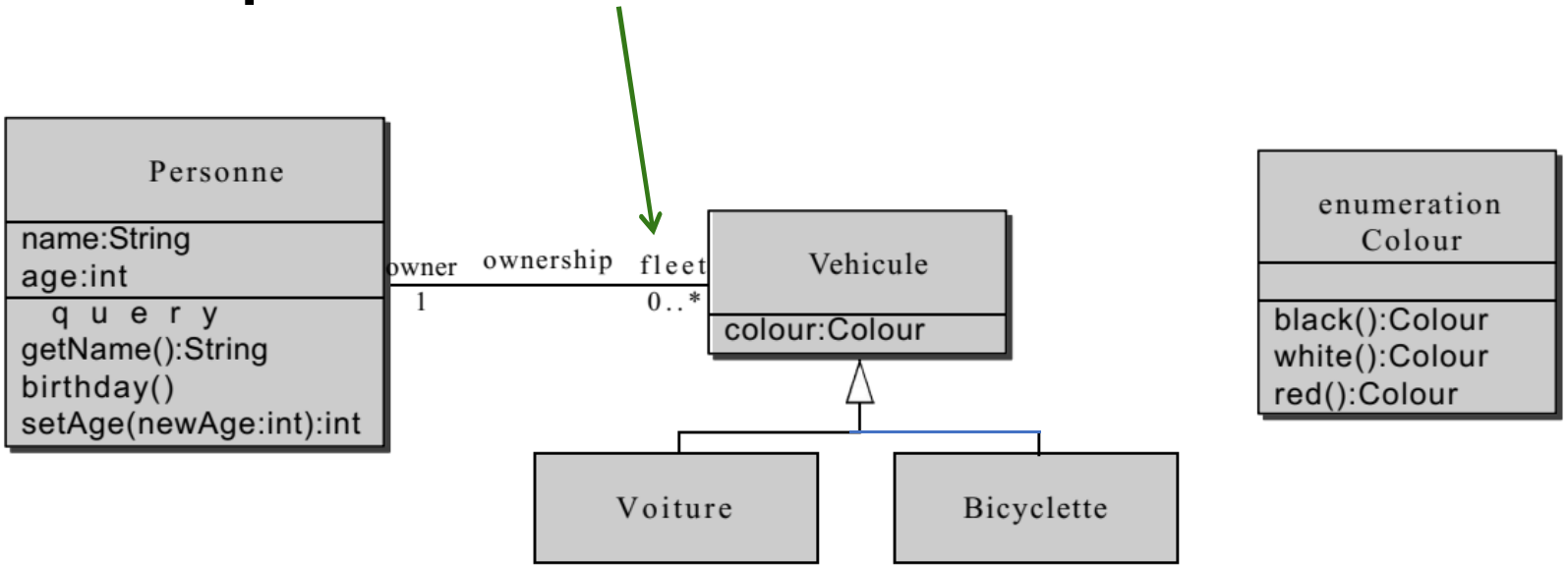


"Le propriétaire (owner) d'un véhicule doit être âgé d'au moins 18 ans"

Context Vehicule
Inv: self.owner. age > = 18

1. Introduction

- **Exemples d'OCL**



"Personne ne possède plus de 5 véhicules"

```
Context   Personne
inv:      self.fleet->size <= 5
```

1. Introduction

- **Alors !**
  - Nous avons besoin d'un langage pour aider avec la spécification.
  - Nous recherchons un **«add-on»** au lieu d'un tout nouveau langage avec des capacités de spécifications complètes.
  - Pourquoi pas la logique du premier ordre? - Pas OO.
  - OCL est utilisé pour spécifier les contraintes sur les systèmes OO.
  - OCL n'est pas le seul.
  - Mais OCL est le seul à être standardisé.

- **Où utiliser OCL?**
  - Spécifier des ***invariants*** pour les classes et les types
  - Spécifier les conditions préalables et postérieures aux méthodes
  - Comme langage de navigation
  - Pour spécifier des contraintes sur les opérations
  - Tester les Exigences et les spécifications

Sommaire

1. Introduction
2. **Typologie des contraintes**
   - ***Notion de Contexte***
   - ***Invariants***
   - ***Pré et post conditions***
   - ***Conception par contrat***
3. Types de base et opérations
4. Accès aux objets et navigation
5. Opérations pour les collections OCL
6. *Conclusion*

2. Typologie des contraintes: **Notion de contexte**

- **Mot-clé context: Une contrainte OCL est toujours définie dans un contexte.**
  - Ce contexte est l'instance d'une classe
- **Exemple:**
  - **context Personne:**

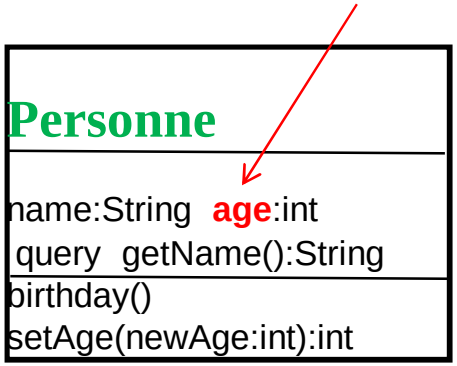
    La contrainte OCL s'applique à la classe **Personne**, c'est-à-dire à toutes les instances de cette classe.

  - **Context Personne::setAge(newAge:int):int**

    La contrainte OCL s'applique à la méthode **setAge(newAge:int).**

2. Typologie des contraintes: Invariants

- **Invariants**
- **Un invariant exprime une contrainte sur un objet ou un groupe d'objets qui doit être respectée en permanence.**
- Un **invariant** est une expression OCL booléenne - prend la valeur true / false.
- **Mot-clé : inv:**
- **Exemple**



context Personne

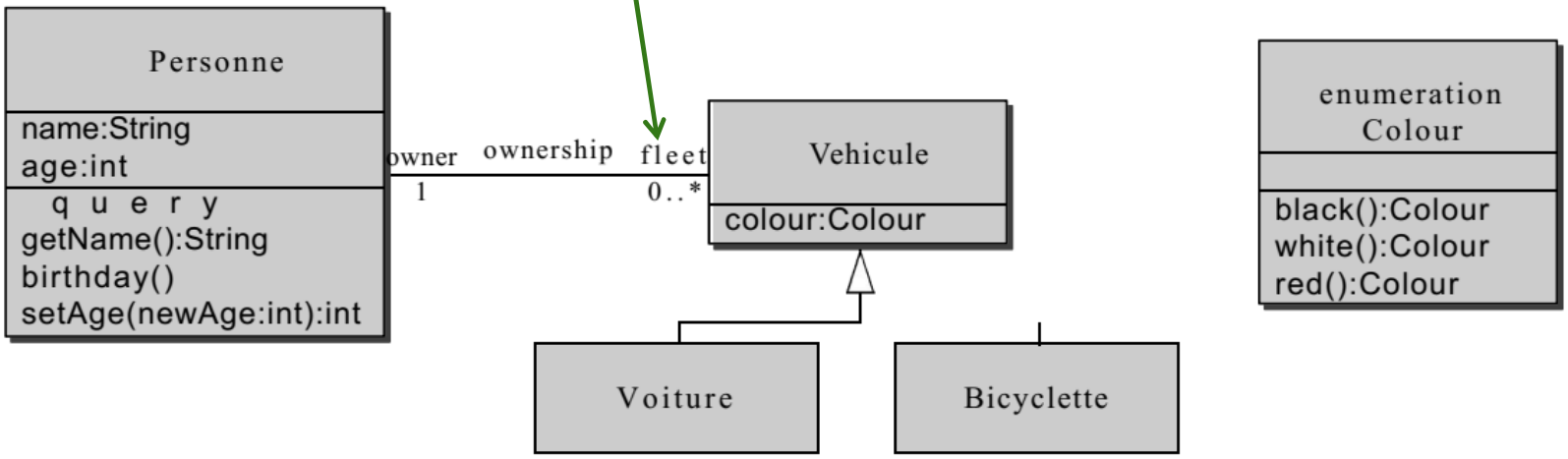
inv: age >= 18

- Pour toutes les instances de la classe **Personne**, l'attribut **age** doit toujours être supérieur ou égal à 18 ans.

2. Typologie des contraintes: Invariants

- Les objets de contexte peuvent être désignés dans l'expression à l'aide du mot-clé «**self**».
- **Exemple**

Context Personne
inv: self.fleet–>size <= 5



Context Personne
inv: self.fleet–>size <= 5

2. Typologie des contraintes: Pré et Post conditions

OCL peut également spécifier des opérations à l'aide des Pré et Postconditions.

- **Précondition :** état qui doit être respecté avant l'appel de l'opération
- **Postcondition :** état qui doit être respecté après l'appel de l'opération
- **Mots-clés : pre** et **post**

Dans la postcondition, deux éléments particuliers sont utilisables

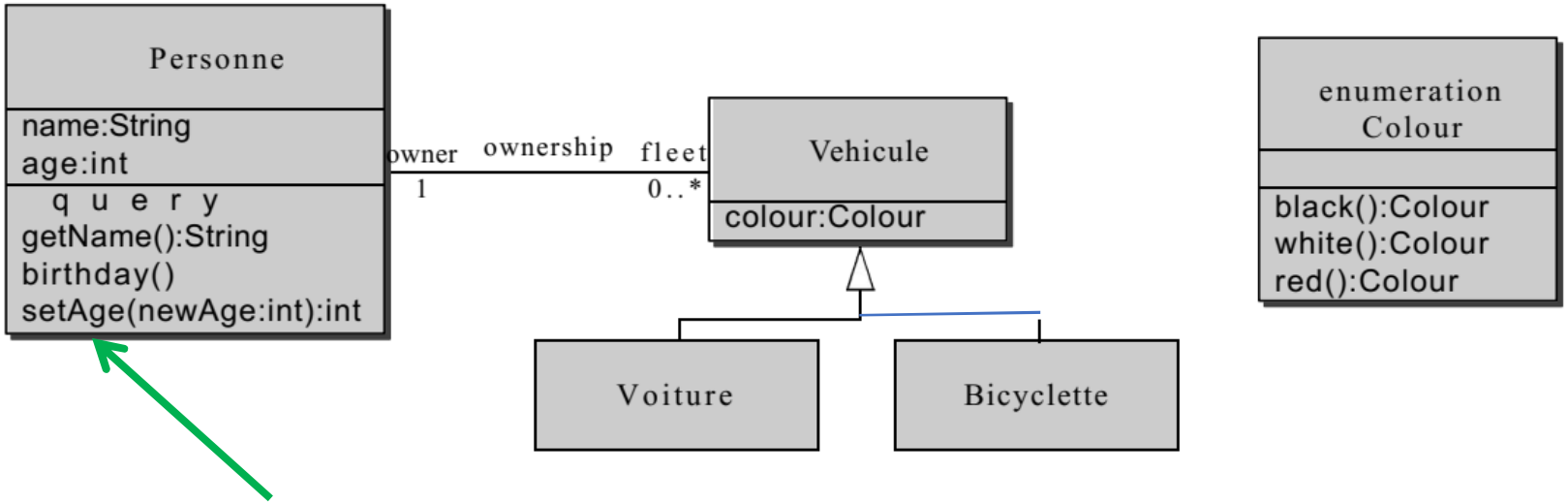
- Attribut **result :** référence la valeur retournée par l'opération
- **mon_attribut@pre** : référence la valeur de *mon_attribut* avant l'éxécution de l'opération.

Syntaxe pour préciser l'opération

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

2. Typologie des contraintes: Pré et Post conditions

- **Exemples de Pré et Postconditions**

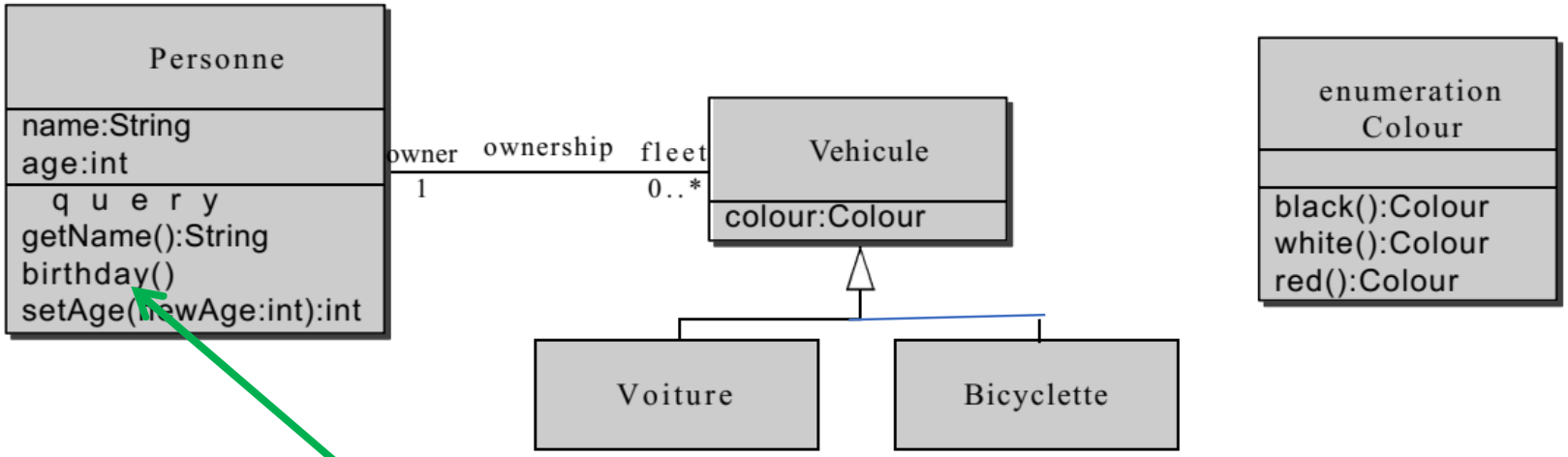


- "Si **setAge** (...) est appelée avec un argument non négatif, l'argument devient la nouvelle valeur de l'attribut age."

Context Person::setAge(newAge:int)
pre: newAge >= 0
Post: self.age = newAge

2. Typologie des contraintes: Pré et Post conditions

- **Exemples de Pré et Postconditions**



"L'appel de birthday () augmente l'âge d'une personne de 1."

context Person::birthday()
Post: self.age = self.age@pre + 1

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

2. Typologie des contraintes: Conception par Contrat

- **Pre et Post conditions permettent de définir une conception par contrat entre l'appelant d'une opération et l'appelé:**
  - Si l'appelant respecte les contraintes de la **pré-condition** alors l'appelé s'engage à respecter la **post-condition.**
  - Si l'appelant **ne respecte** pas la **pré-condition**, alors le résultat de l'appel est **indéfini.**
- **Dans l'exemple précédent:**

```
Context Person::setAge(newAge:int)
pre:      newAge >= 0
Post:     self.age = newAge
```

  - Si l'appelant à **setAge** respecte la pré-condition **newAge >=0**, alors **setAge** affectera à l'attribut **age** la valeur du parametre **newAge**.
  - Sinon, la valeur de l'attribut **age** sera indéfinie.

Sommaire

1. Introduction

2. Typologie des contraintes

3. **Types de base et opérations**

4. Accès aux objets et navigation

5. Opérations pour les collections OCL

6. *Conclusion*

3. Types de base et opérations : ocl langage fortement typé

- Pour être bien formée, une expression OCL doit être conforme aux règles de conformité de type.
  - Par exemple, vous ne pouvez pas comparer un **entier** avec une **chaîne de caractères**.
- Chaque classificateur dans un modèle UML devient un **type OCL**
  - Par exemple. chaque **classe** que vous créez
- Types et opérateurs prédéfinis dans les contraintes OCL:

Boolean -> true, false

- **Opérations**: and, or, xor, not, implies, if-then-else

Integer -> 1, -5, 2, 489, 26524, ...

- **Opérations** : *, +, -, /, abs()

Real -> 1.5, 3.14, ...

- **Opérations** : *, +, -, /, floor()

String -> 'Université Mila...'

- **Opérations** : toUpper(), concat()

3. Types de base et opérations:Ensembles, Bags, Sequences

- **Les ensembles, Multi-Ensembles (Bags) et séquences:**
  - sont prédéfinis dans OCL et sont des sous-types de **collection**.
  - OCL propose un grand nombre d'opérations **prédéfinies** sur les collections. Ils sont tous de la forme:

    collection-> opération (arguments)

- **La collection OCL-Type est la superclasse générique d'une collection d'objets de type T**
- **Les sous-classes de collection sont:**
  - **Set**: Défini au sens mathématique. Chaque élément ne peut apparaître qu'une seule fois
  - **Bag**: une collection, dans laquelle les éléments peuvent apparaître plusieurs fois (multi-Set)
  - **Séquence**: un multi-ensemble, dans lequel les éléments sont ordonnés

- **Exemples :**
  - **{ 1, 4, 3, 5 } : Set**
  - **{ 1, 4, 1, 3, 5, 4 } : Bag**
  - **{ 1, 1, 3, 4, 4, 5 } : Sequence**

Sommaire

1. Introduction
2. Typologie des contraintes
3. Types de base et opérations
4. **Accès aux objets et navigation**
5. Opérations pour les collections OCL
6. *Conclusion*

4. Accès aux Objets et Navigation

- Dans une contrainte OCL associée à un objet, on peut:
  - **Accéder** à l'état interne de cet objet (**ses attributs**)
  - **Accéder** aux opérations de cet objet : **self.operation.**
  - **Naviguer** dans le diagramme : accéder de manière **transitive** à tous les objets (et leur état) avec qui il est en relation

4. Accès aux Objets et Navigation: Exemple

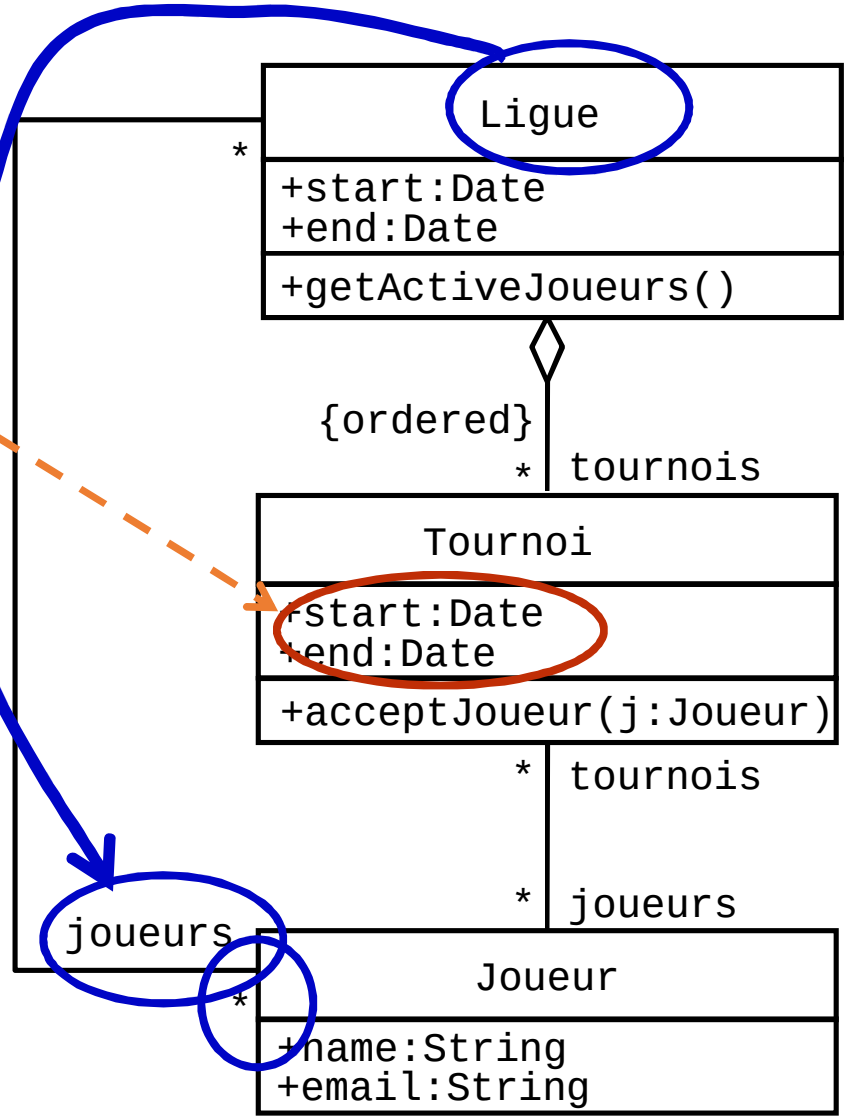
- **Navigation Locale ----- > attributs**

```
context Tournoi inv:

  end - start <= 7
```

- **Navigation Directe relative Aux classes**

```
context
 Tournoi::acceptJoueur(j)
  pre:
   ligue.joueurs->includes(j)
```

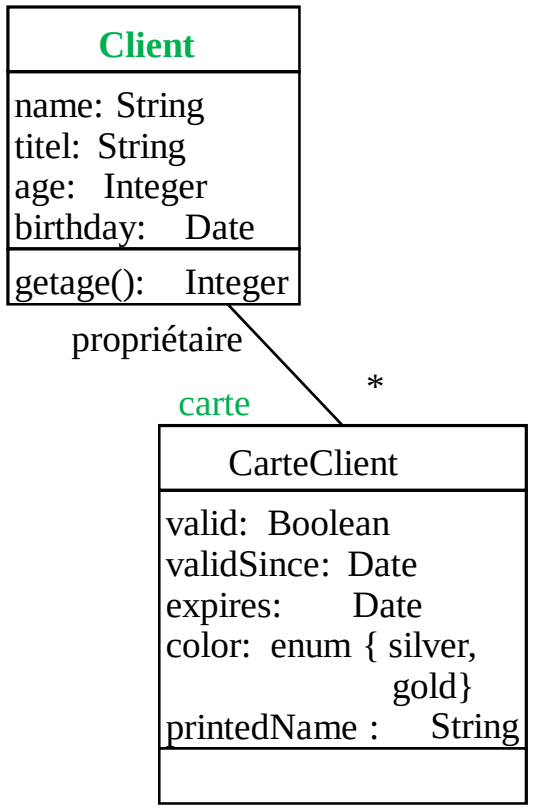


4. Accès aux Objets et Navigation: Association 1:n

- **Exemple: un client ne doit pas avoir plus de 3 cartes**

```
context Client inv:
carte->size <= 3
```

carte désigne un ensemble de cartes de clients



4. Accès aux Objets et Navigation: plusieurs Associations 1:r

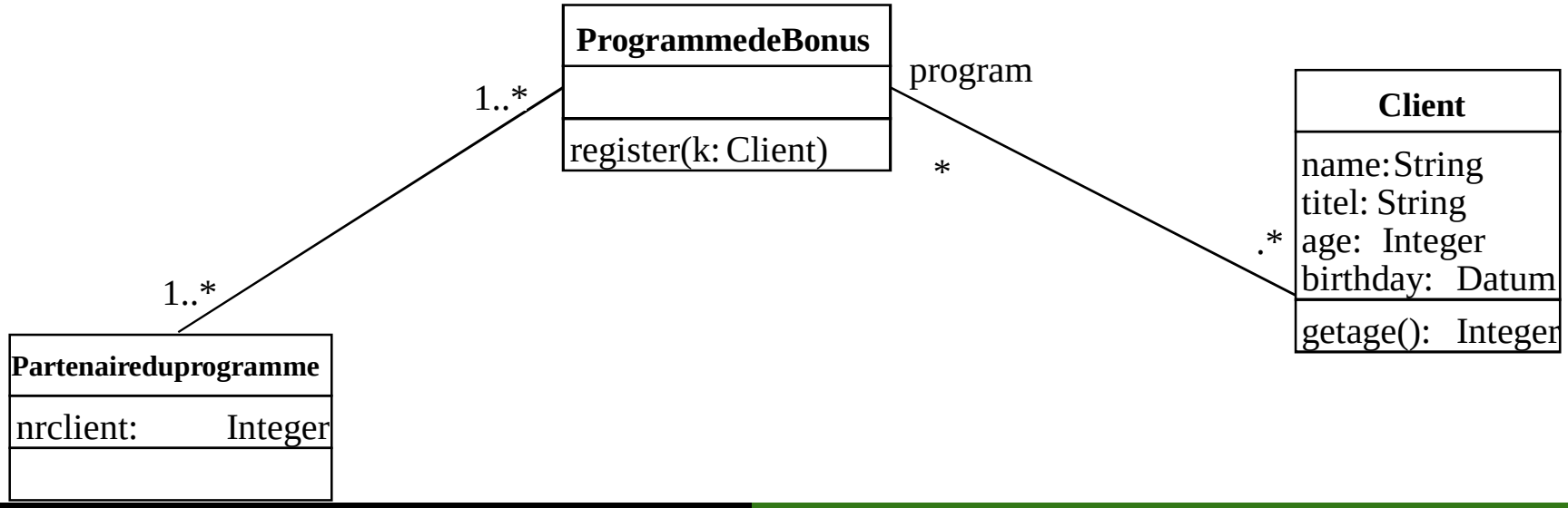
Exemple:

Context Partenaireduprogramme

```
nrclient = programmedebonus.client->size
```

`Client` denote un multi-ensemble de `client`

`programmedebonus` denote un ensemble de `programmedebonus`

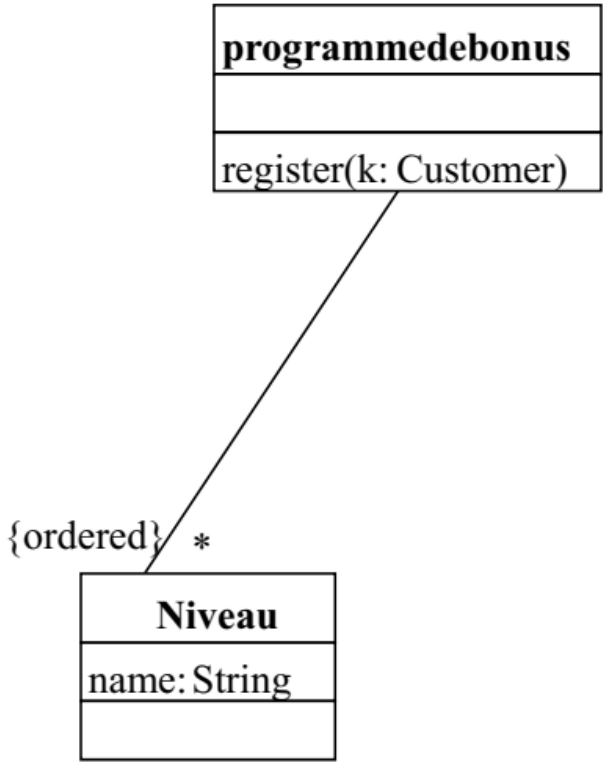


4. Accès aux Objets et Navigation: association avec contrainte

- **La navigation à travers une association avec la contrainte {ordered} produit une séquence.**
- **Exemple**

```
context programmedebonus
niveau->size = 2
```

`niveau` denote une sequence de `niveaux`



Sommaire

1. Introduction
2. Typologie des contraintes
3. Types de base et opérations
4. Accès aux objets et navigation
5. **Opérations pour les collections OCL**
6. *Conclusion*

5. Opérations pour les collections OCL

Les collections sont tous de la forme: **collection-> opération (arguments)**

`size: Integer`

Retourne le nombre d'éléments dans la collection

`includes(o:OclAny): Boolean`

Retourne True, si l'élément o est dans la collection

`count(o:OclAny): Integer`

Compte le nombre de fois qu'un élément est contenu dans la collection

`isEmpty: Boolean`

Retourne True, si la collection est vide

`notEmpty: Boolean`

Retourne True, si la collection n'est pas vide

Le type OCL `OclAny` est le type OCL le plus général

5. Opérations pour les collections OCL (Suite)

`union(c1:Collection)`

Retourne l'union avec la collection `c1`

`intersection(c2:Collection)`

Retourne l'intersection avec Collection `c2` (contient uniquement des éléments, qui apparaissent dans la collection ainsi que dans la collection c2)

`including(o:OclAny)`

Collection contenant tous les éléments de la collection et l'élément o

`select(expr:OclExpression)`

Sous-ensemble de tous les éléments de la collection, pour lesquels l'expression OCL expr est vraie

5. Opérations pour les collections OCL (Suite)

- **Comment obtenir les collections OCL?**

- **Une collection peut être générée en énumérant explicitement les éléments**
- **Une collection peut être générée en naviguant le long d'une ou plusieurs associations 1:N**
  - La navigation le long d'une **seule association** 1: n donne un ensemble (**set**)
  - La navigation le long de **quelques associations** 1: n donne un **Bag** (multi-ensemble)
  - La navigation le long d'une seule association 1: n étiquetée avec la contrainte {ordonnée} donne une séquence (**sequence**)

6. Conclusion: OCL Object Constraint Language

- **Langage de contraintes orienté-objet.**
- **Langage formel (mais « simple » à utiliser) avec une syntaxe, une grammaire, une sémantique.**
- **S'applique entre autres sur les diagrammes UML.**
- **Outils :**
  - ✓ **USE (Mark Richters) UML-based Specification Environment,** http ://sourceforge.net/projects/useocl
  - ✓ **Octopus (Warmer & Kleppe),** http ://octopus.sourceforge.net/
  - ✓ **Dresden OCL ,** http ://dresden-ocl.sourceforge.net/

Références

- **Allaoua chaoui,**

  “ Introduction à OCL, Université de Constantine 2"
- **Jos Warmer and Anneke Kleppe**,

  “The Objection Constraint Language: Precise Modeling with UML”, by Jos Warmer and Anneke Kleppe
- **Eric Cariou**,

  " Object Constraint Language" , Université de Pau-France
- **Marianne Huchard**,

  "Object Constraint Language (OCL) *Une introduction"*
- **Bernhard Beckert ,**

  “ Introduction to OCL, Universität Koblen Landau"
- **Laetitia Matignon,**

" **OCL,** Université Claude Bernard Lyon 1 "