Analyse, Conception et Programmation Orientées Objets

–

Model Driven Development

Applications en UML 2.0 et Java

Bruno Mermet

2011

Plan Général

- Les concepts Objet
- Analyse et Conception Orientées Objets en UML

Les concepts Objet

Plan Général

- Du typage à la notion de classe
- Les relations d’héritage et de généralisation
- Les liens entre les classes
- La généricité

Donnée de base

- Description
  - un type : entier, chaîne de caractères, réel, etc.
  - une valeur (12, « mon texte », 7.28)

Type d'une donnée = Ensemble des valeurs possibles pour cette donnée

- Inconvénients
  - ensemble de types limité
  - **typage statique**
  - pas de notion de sous-typage
  - aspect statique seul abordé

Les enregistrements

- Principe :
  - définir des données comme produits cartésiens d'autres données (exemple : entier × entier × chaîne)
  - un **enregistrement** = un n-uplet
    - exemples :
      - (10, 20, « rose ») caractérise un point de l'écran
      - ((« Durand », « Paul »), (« Durand », « Anne »)) désigne un couple
- Réutilisation :
  - utilisation d'un constructeur de type *enregistrement*
    - un nom pour le type
    - ensemble de champs nommés et typés

Enregistrements : exemple

- Introduction d'un type « livre » :

  livre {
  titre : chaîne de caractères,
  auteur : chaîne de caractères,
  éditeur : chaîne de caractères,
  année de parution : date,
  nombre de page : entier,
  prix : réel
  }

- Utilisation

  si *RéfProlog* donnée de type *livre*, alors *RéfProlog.titre* désigne son titre (et vaut « The Art of Prolog »)

Enregistrements : bilan

- Ensemble des types :
  - extensible à volonté
- Typage :
  - toujours statique
  - toujours pas de notion de sous-type

$\Rightarrow$ Passage au concept de classe et d'objet

Notions de classe et d'objet

- Comparaison
  - Type ⇒ **Classe**
  - Donnée ⇒ **Objet** (**instance** d'une classe)
  - Champ ⇒ Champ(*)

- Exemple :
  - RéfProlog serait un objet de classe Livre

(*) ou attribut, ou variable d'instance

Classes et types : premières différences

- Type :
  - point de vue statique ⇒ les champs
- Classe :
  - point de vue statique : état
    - les champs (avec valeur par défaut éventuelle)
    - les propriétés (ou contraintes)
  - point de vue dynamique : changements d'état
    - les **méthodes** (définies au niveau de la classe, mais s'appliquent sur les objets de cette classe), avec d'éventuelles **pré-conditions** et **post-conditions**.

Classes : premier exemple

Personne
Nom : Chaîne de caractères Prénom : Chaîne de caractères Date de naissance : Date Situation de famille : Chaîne de caractères = « Célibataire » Sexe : Chaîne de caractères
Créer (nom, prénom, date, sexe) Créer (nom, prénom, date, sexe, situation) ModifierSituation (NouvelleSituation) **Surcharge**
Sexe = « m » ou Sexe = « f » Nom ≠ « »

N.B. : si Toto objet de classe Personne, Modifier la situation de Toto se fait par Toto.ModifierSituation(« Marié »)

Vie et mort d'un objet

- Un objet :
  - nait
  - vit
  - meurt
- Par classe, 3 types de méthodes :
  - **constructeurs**
  - **destructeurs**
  - …les autres !

⇒ Rien n'est possible sur un objet non créé

⇒ Rien n'est possible sur un objet détruit

Persistance d'un objet

- Mort d'un objet classique :
  - appel de son destructeur ;
  - arrêt du système qui l'a créé.
- Inconvénient :
  - pannes ;
  - maintenance ;
  - évolution.

$\Rightarrow$ Notion d'objet persistant

- Un **objet persistant** est un objet dont la vie continue après l'arrêt du système l'ayant créé

Variables et méthodes de classe

- Cas général :
  - champ : propre à un objet
  - impossibilité à des objets d'une même classe de partager des données

⇒ Variable de classe :

  - même donnée pour tout les objets de la classe

- Méthodes de classe :
  - ne s'appliquent pas particulièrement à un objet

  ⇒ à voir comme des fonctions de bibliothèques

Vers la réutilisabilité

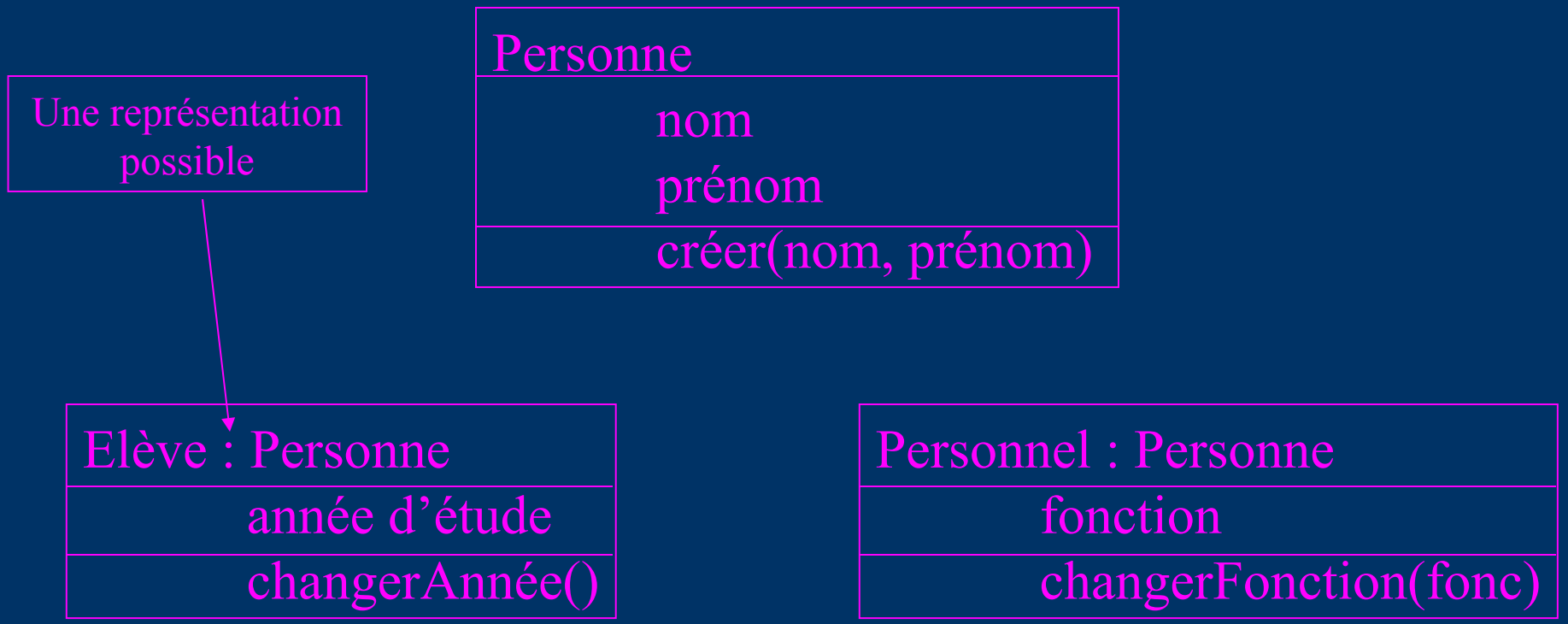
Elève
nom prénom
année d'étude
créer(nom, prénom)
changerAnnée()

Personnel
nom prénom
fonction
créer(nom,prénom)
changerFonction(fonc)

⇒ Comment factoriser les parties communes ?

Héritage : introduction

Une représentation possible



- Vocabulaire :

  Elève **hérite**/est une **sous-classe**/est une **spécialisation** de Personne

  Personne est une **généralisation**/**super-classe** d'Elève

  On emploi aussi les termes de classe mère/classe fille

Héritage et sous-typage

- Autre lecture de la relation d'héritage :
  - un objet d'une classe fille *est-un* objet de la classe mère

⇒ Tout objet d'un type d'une classe fille peut être utilisé à la place d'un objet du type d'une classe mère

- Sur l'exemple : *créer* peut s'appliquer pour un élève ou pour le personnel

La réciproque n'est pas vraie !

Redéfinition et liaison dynamique

- Principe de la **liaison dynamique** :

Quadrilatère
l1, l2, l3, l4 : réels
Périmètre() {l1+l2+l3+l4} créer(p1, p2, p3, p4)

Parallélogramme : Quadrilatère

Losange : Quadrilatère
Périmètre() {4 x l1}

la méthode *périmètre* de la classe *Quadrilatère* est **redéfinie** dans *Losange*

P de type Parallélogramme, L de type Losange, Q de type Quadrilatère :

Q := P, Q.périmètre() calcule *l1+l2+l3+l4*

Q := L, Q.périmètre() calcule $4 \times l1$

Classes et méthodes abstraites

- Une **classe abstraite** *C* est une classe dont les seuls objets qui peuvent en être membres sont membres d'une sous-classe de *C*.

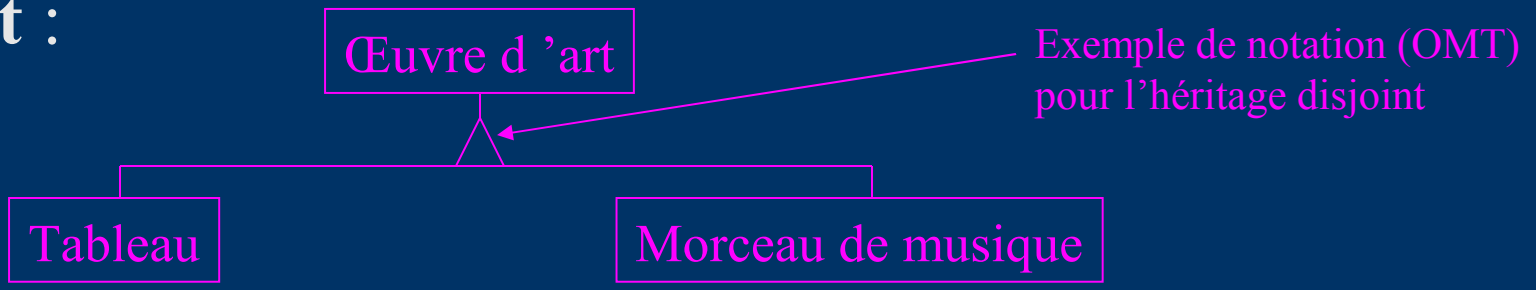
  Corollaire : il n'est pas possible de créer un objet de classe *C* si *C* est une classe abstraite.

- Une **méthode abstraite** est une méthode dont la définition n'est pas donnée dans la classe courante, mais devra l'être dans des classes filles.

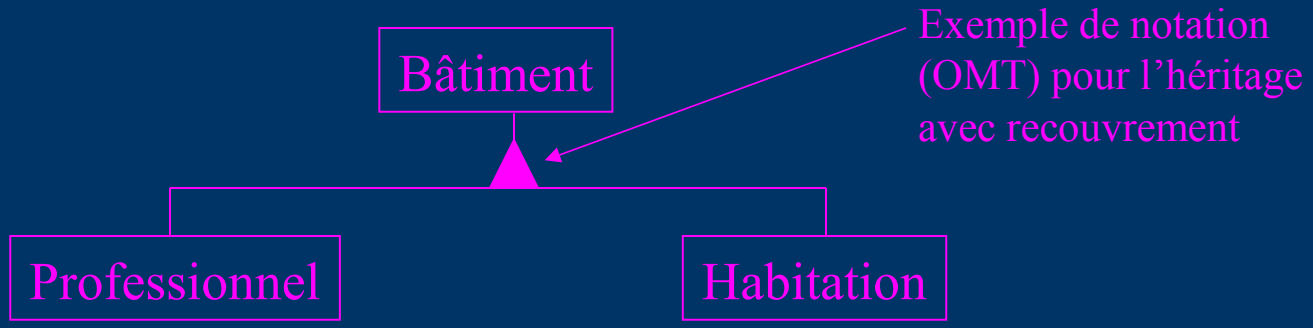
  Toute classe contenant au moins une méthode abstraite est une classe abstraite.

Héritage : disjonction ou recouvrement

- Si un même objet ne peut pas être membre de plusieurs sous-classes d'une même classe, on parle d'héritage **disjoint** :

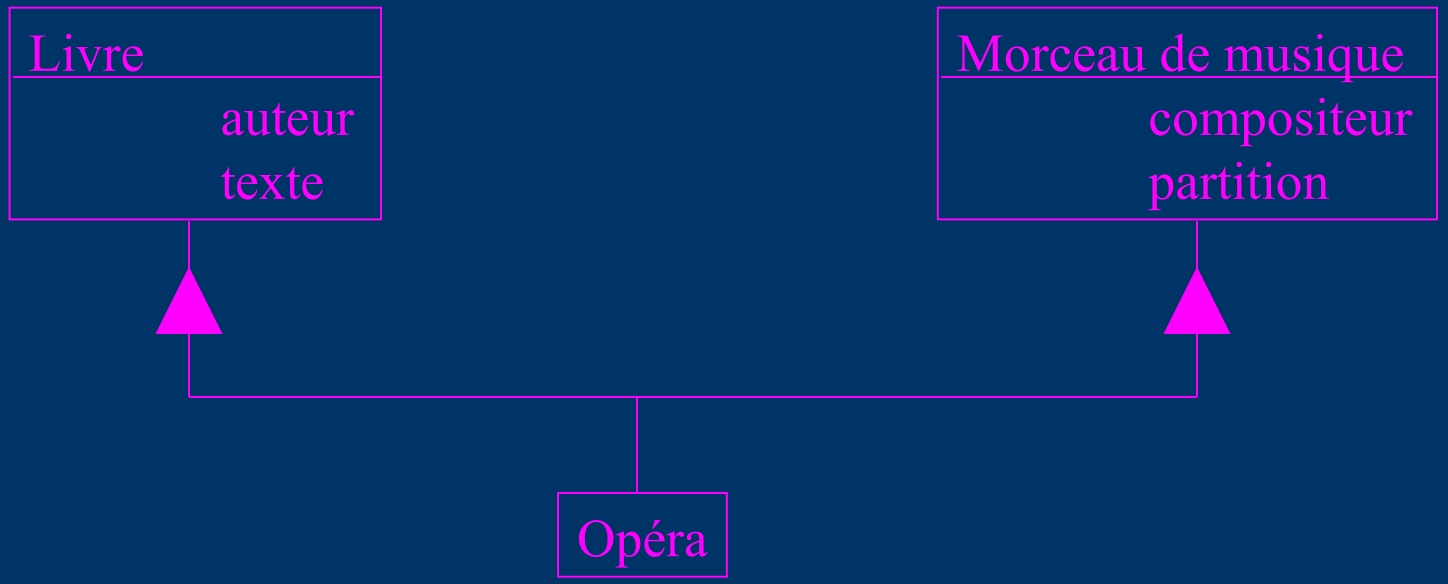


- Sinon, on parle d'héritage **avec recouvrement** :



Héritage multiple

- Il y a **héritage multiple** si une classe peut hériter de plusieurs autres classes :



Héritages : propriétés

- Transitivité :

  Si *A* hérite de *B* et si *B* hérite de *C*, alors *A* hérite de *C*

  Donc si tout objet de *A* *est-un* objet de *B* et tout objet de *B* *est-un* objet de *C*, alors tout objet de *A* *est-un* objet de *C* (transitivité)

- Réflexivité

  Une classe *C* ne peut pas hériter d'elle-même ; par contre un objet instance de *C* *est-un* objet de *C*

- Non Symétrique/Antisymétrie

  Si *A* hérite de *B*, *B* n'hérite pas (et ne peut être déclarée comme héritant) de *A*

  Par contre, si tout objet de *A* *est-un* objet de *B* et inversement, alors *A* = *B* (antisymétrie)

- Corollaire

  Un cycle dans la relation d'héritage n'est pas possible

  $\Rightarrow$ les relations d'héritage d'un système forment un ensemble de **graphes acycliques directs** (DAG)

  $\Rightarrow$ La relation *est-un* est une **relation d'ordre**

Classes et méthodes terminales

- Une **classe terminale** est une classe dont aucune classe ne peut hériter
- Une **méthode terminale** est une méthode qui ne peut pas être redéfinie

$\Rightarrow$ Usage à réserver à des cas bien particuliers

Liens entre classe (1)

Comment :

- Par les champs

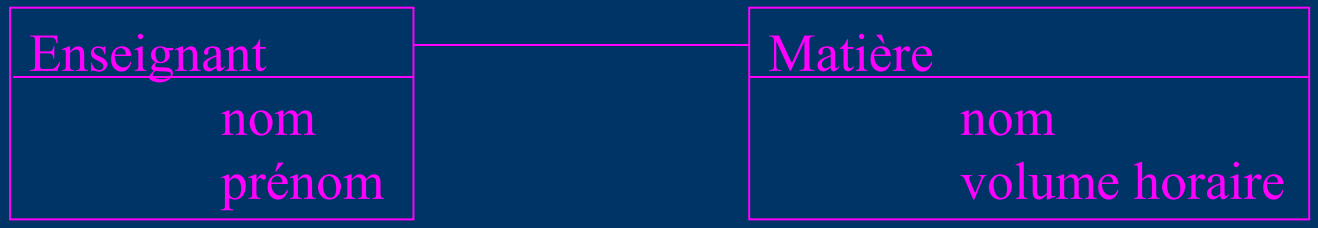
  en déclarant un champ de classe A dans une classe B

- Dans les méthodes

  en déclarant une variable locale de classe A dans une méthode de la classe B

- Par la définition explicite d'une relation

  exemple :



Agir sur un objet d'une classe liée

- Actions possibles
  - Lire la valeur d'un champ :

    destination := nom_objet.nom_champ
  - Modifier un champ :

    nom_objet.nom_champ := nouvelle_valeur
  - Exécuter une méthode :

    [résultat :=] nom_objet.nom_méthode ([param d'entrée])
- Remarques :
  - symbole d'affection (:=) dépend de la notation
  - le texte entre crochets est optionnel

Actions sur objets : constat

- Rôle des méthodes :
  - factoriser du code commun ;
  - Avoir un typage plus fort (ne s'applique que sur les objets de la classe *C* dans laquelle elle est définie ou d 'une classe héritant de *C*);
  - n'autoriser que certains changements d'état sur un objet.
- Mais
  - possibilité d'intervenir directement depuis l'« extérieur » sur un champ d'une méthode annule le dernier point

Gestion des droits d'accès

- Conséquence du constat précédent :

  Pour chaque champ et chaque méthode, possibilité de définir des droits d'accès pour les autres classes :

droit	dans *C*	dans le *package*	dans les classes filles	dans les autres classes
public	oui	oui	oui	oui
protégé	oui	oui	oui	non
paquetage	oui	oui/non	non	non
privé	oui	non	non	non

Assouplir et mieux contrôler les droits (C++)

- Assouplir : création de la notion d'**ami** :
  - Si *A* est amie de *C*, alors *A* a la même visibilité que *C* sur les champs et méthodes
- Augmenter le contrôle :
  - différents types d'héritage : public, privé, protégé :

Nouveaux types dans la classe fille selon le type d'héritage			
Type d'héritage \ Type de départ	public	protégé	privé
public	public	protégé	
protégé	protégé	protégé	
privé	privé	privé	

- Rmq : En Java, paquetages ≈ classes amies

Encapsulation (1)

- Définition :

  Modifier les champs d'un objet n'est possible que par l'intermédiaire des méthodes définies directement ou par héritage dans la classe de cette objet

- Principe :
  - Les champs sont tous privés
  - Les méthodes mises à la disposition des autres classes sont déclarées *publiques*

  N.B. : si classe à vocation à être utilisée comme classe mère, certains champs et méthodes pourront être *protégés*

Encapsulation (2)

- Avantages :
  - respect du troisième rôle des méthodes (n'autoriser que certains changements d'état pour un objet)
  - possibilité de modifier la représentation interne d'une classe (les champs) sans devoir refaire tous les projet l'utilisant
- Conséquence pour diffuser au public une classe

  préciser
  - nom et rôle de la classe ;
  - profils et rôles des différentes méthodes publiques
  - si nécessaire, rôle des champs et méthodes protégés

  = **partie publique** (par opposition à la **partie privée**)

Réutilisation des structures de données

- Problème
  - liste d'entiers
    - ajouter un élément (entier) ;
    - enlever le premier élément ;
    - donner la longueur de la liste ;
    - trier les entiers de la liste.
    - …
  - liste de chaînes de caractères :
    - ajouter un élément (chaînes) ;
    - enlever le premier élément ;
    - donner la longueur de la liste ;
    - trier les chaînes de la liste.
    - …

Parties communes

Parties différentes

⇒ Héritage inutilisable

Interface

- Définition
  - Une **interface** est un ensemble nommé de profils de méthodes
- Implantation
  - Une classe implante une interface si elle :
    - le déclare
    - implante toutes les méthodes spécifiées dans l'interface
- Utilisation
  - Si une classe est reliée à un type d'interface donné, elle peut être reliée à toute classe implantant cette interface.

Généricité

- Principe : paramétrer une classe par une autre

Liste <Elément>	Entier
Ajouter un élément Enlever le premier élément Calculer la longueur de la liste	

⇒ Définition d ’un objet : A : Liste<Entier>

- **Généricité contrainte** :

  But : permettre l’utilisation d’une fonction de la classe paramètre

Généricité contrainte : exemple

Liste <Elément→ordre>
ajouter un élément supprimer le dernier élément calculer la longueur de la liste ordonner la liste = faire un tri à bulle en utilisant la méthode *comparer* définie sur les éléments

Booléen
x : booléen

Entier
x : Entier bool comparer(y : entier)

Chaîne
x : Chaîne bool comparer(y : chaîne)

- A : Liste<Booléen> : impossible
- B : Liste<Entier> : possible ; B.ordonner() trie numériquement
- C : Liste<Chaîne> : possible ; C.ordonner() trie alphabétiquement