Programmation Orientée Objets

Licence Pro
IUT du Havre
B. Mermet / P. Person

Plan du cours

- Les concepts Objet (BM)
- Mise en œuvre en Java (PS)
- Héritage et polymorphisme (BM)
- Java : packages java.lang et java.util (PS)

Les concepts objet

Notion d'objet

- Principe de base
  - structurer le code d'un programme autour des types de données manipulés
- Vocabulaire
  - Mis à part quelques *types de base* (différents selon les langages)
    - Un type de donnée = une *classe*
    - Une donnée = un *objet* *o* d'un type donné *t*. On dit que *o est instance de t* (où que *o est de type t*)

Types de base en java

- Essentiellement des types numériques :
  - int, long, float, double, byte
- Type booléen
  - boolean
- Type caractère
  - char

Classe

- Une classe définit et nomme un type de donnée. Elle précise notamment :
  - La structure des données de ce type ;
    Les *champs*/*attributs*/*variables d'instance*
  - Les façons dont peut évoluer chaque donnée de ce type ;
    Les *méthodes (d'instance)*
  - Comment créer des données de ce type ;
    Les *constructeurs*
  - (Comment détruire les données de ce type).
    Les *destructeurs*
- Convention
  - Les noms des classes commencent par une majuscule

Variable d'instance

- Une variable d'instance d'une classe C est définie essentiellement par un triplet (visibilité[1], type, nom). Il s'agit d'une caractéristique qui sera propre à chaque objet instance de C.

- Exemple
  - La classe *Personne* peut définir les variables d'instance suivante
    - (..., String, nom)
    - (..., int, âge)
  - Chaque *personne* (objet instance de la classe *Personne*) pourra avoir son propre nom et son âge. Plusieurs personnes peuvent avoir le même nom, le même âge, voire le même nom et le même âge.

[1] : voir plus tard

Méthodes (d'instance) : principes

- Une méthode définit comment un objet peut évoluer et/ou comment obtenir une information sur un objet. Elle est définie par un tuple (visibilité[1], type de retour, nom, listeDeParamètres, corps).
- Le *type de retour* est le type de l'information retournée, la *liste de paramètres* est une liste de couple (type, nom) spécifiant quelles informations l'utilisateur de la méthode va fournir et le *corps* décrit ce que fait la méthode.
- Une méthode définie dans une classe C sera toujours utilisée en l'appliquant à un objet instance de C.
- Dans le corps d'une méthode, les variables d'instance de l'objet auquel la méthode est appliquée peuvent être directement utilisées.
- Remarque : plusieurs méthodes de même nom peuvent coexister à partir du moment où leurs listes des types de leurs paramètres diffèrent)

[1] : voir plus tard

Méthodes d'instance : exemples

- Accesseurs
  - Accesseurs en lecture (*getters*)

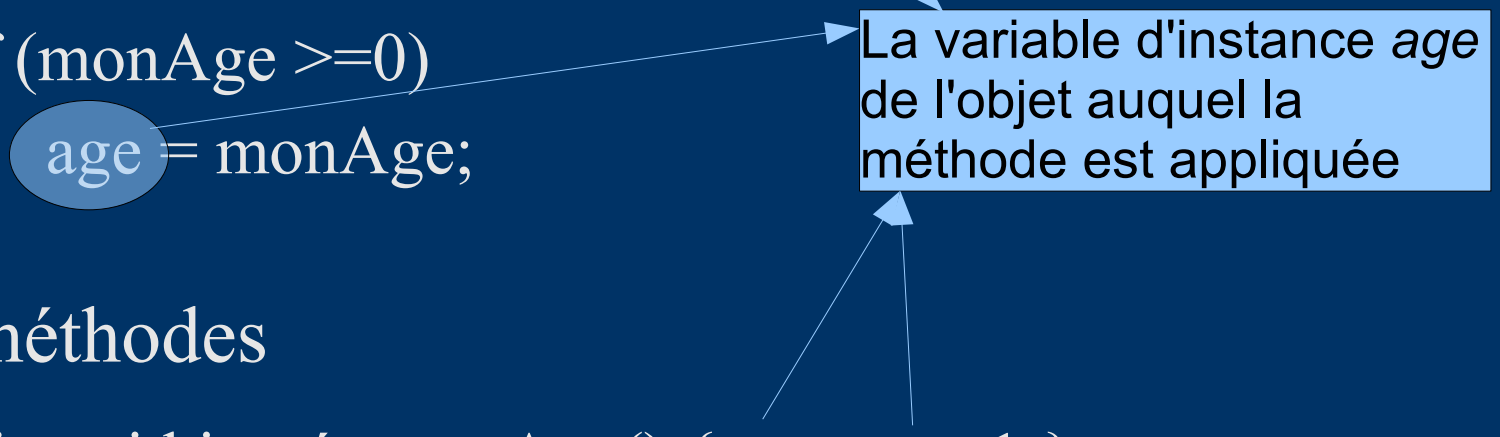
```
public int getAge() {return age;}
```

  - Accesseurs en écriture (*setters*)

```
public void setAge(int monAge) {
    if (monAge >=0)
        age = monAge;
    }
```

- Autres méthodes

```
public void incrémenteAge() {age=age+1;}
```



Constructeurs

- Un constructeur est un code nommé et paramétré associé à une classe définissant comment les objets de la classe sont initialisés.
- En java, un constructeur
  - N'a pas de type de retour
  - A pour nom le nom nom de la classe dont il est constructeurs
  - Plusieurs constructeurs peuvent exister pour une même classe si la leurs listes de types de paramètres diffèrent
- Constructeur par défaut en java
  - Dans toute classe où l'utilisateur n'a définit aucun constructeur (et seulement dans ce cas), un constructeur par défaut est automatiquement ajouté. Ce constructeur ne prend aucun paramètre et initialise chacune des variables d'instance à sa valeur par défaut.

Constructeurs : exemples

```
Public Personne() {
    Nom = "";
    Age = 20;
}

Public Personne(String monNom, int monAge) {
    nom = monNom;
    age = monAge;
}

public Personne(String monNom) {
    nom = monNom;
    age = 30;
}
```

Destructeurs

- Un destructeur est un code nommé et paramétré associé à une classe définissant comment les objets de la classe sont détruits.
- En java, un seul destructeur peut éventuellement être défini :
  - Il s'appelle finalize()
  - Il n'a pas de paramètres

Création d'un objet (en java)

- Créer un objet = utilisation d'une instruction spéciale (new) conjointement avec un constructeur (pour initialiser l'objet).
- Exemples

  New Personne();

  New Personne("toto");

  New Personne("toto",10);
- Rôle de l'instruction de création (new)
  - Réserver de la place en mémoire pour stocker l'objet
  - Retourner la référence[1] à l'objet créé

Donnée et référence

- Notion de référence
  - La référence à une donnée est un identifiant permettant au système de déterminer de manière univoque une donnée. En pratique, il s'agit de l'adresse mémoire où la donnée est stockée.
- En java
  - Les types simples sont gérés directement (on manipule la donnée). Les types objets sont par contre gérés par référence. On n'utilise jamais directement un objet mais toujours sa référence
  - Par simplification d'écriture, partout où une référence sur un type T est utilisé, on note directement T.
  - L'instruction *new* renvoie la référence de l'objet créé.

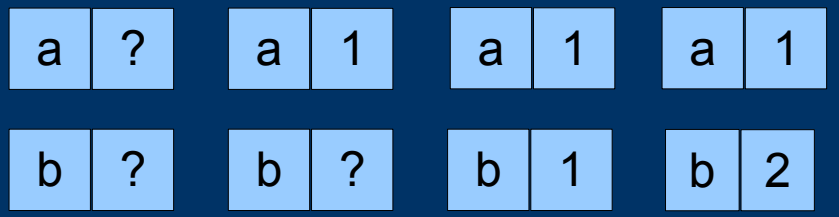
Variable

- Une variable est un couple (nom, valeur) où *valeur* est :
  - une donnée pour les données gérées directement (types simples en Java)
  - Une référence à une donnée pour les données gérées par référence (types objets en Java)
- Utilisation
  - À gauche d'une instruction d'affectation ("=" en Java) pour stocker une valeur dans la variable
  - À droite d'une instruction d'affectation pour récupérer la valeur de la variable)
  - Avant un signe "." pour appliquer une méthode à l'objet dont la référence est contenue dans la variable
  -

Références ou non

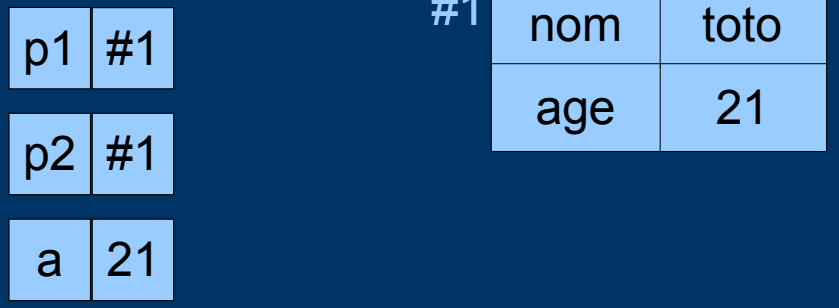
- Types simples

```
int a, b;
a = 1;
b = a;
b = b+1;
```



- Types objets

```
Personne p1,p2;
int a;
p1 = new Personne("toto",20);
p2 = p1;
p2.incrAge();
a = p1.getAge();
```



- L'affectation p2=p1 copie les références, pas les données !

Copier des objets

- Constructeur de copie
  - Principe

    Dans une classe C, on définit un *constructeur de copie* : il s'agit d'un constructeur prenant en paramètre un seul objet o de type C et qui affecte comme valeur à chacune des variables d'instance du nouvel objet une copie de la valeur de la même variable pour l'objet o.
  - Exemple

```
Personne (Personne p) {
  nom = new String(p.nom);
  age = age;
}
```

- Méthode *clone* : non présentée ici

Destruction d'un objet

- Un objet peut être détruit soit explicitement soit automatiquement.
- En Java, les objets sont détruits automatiquement par un processus indépendant, le ramasse-miettes (ou garbage collector).
- Un objet ne peut être détruit que lorsqu'il n'est plus accessible depuis le programme (plus aucune référence ne permet d'y accéder)
- Le ramasse-miettes détruit un objet lorsqu'il juge le moment opportun (il a du temps, la mémoire libre se réduit sérieusement...)
- Le destructeur est appelé par le ramasse-miettes juste avant qu'il ne détruise l'objet

Visibilité

- Concerne
  - Variables d'instance
  - Méthodes
  - Constructeurs
- 4 types

nom	Mot-clef	symbole
publique	public	+
protégée	protected	#
paquetage		
privée	private	-

Accessible depuis n'importe quelle classe

Voir plus tard

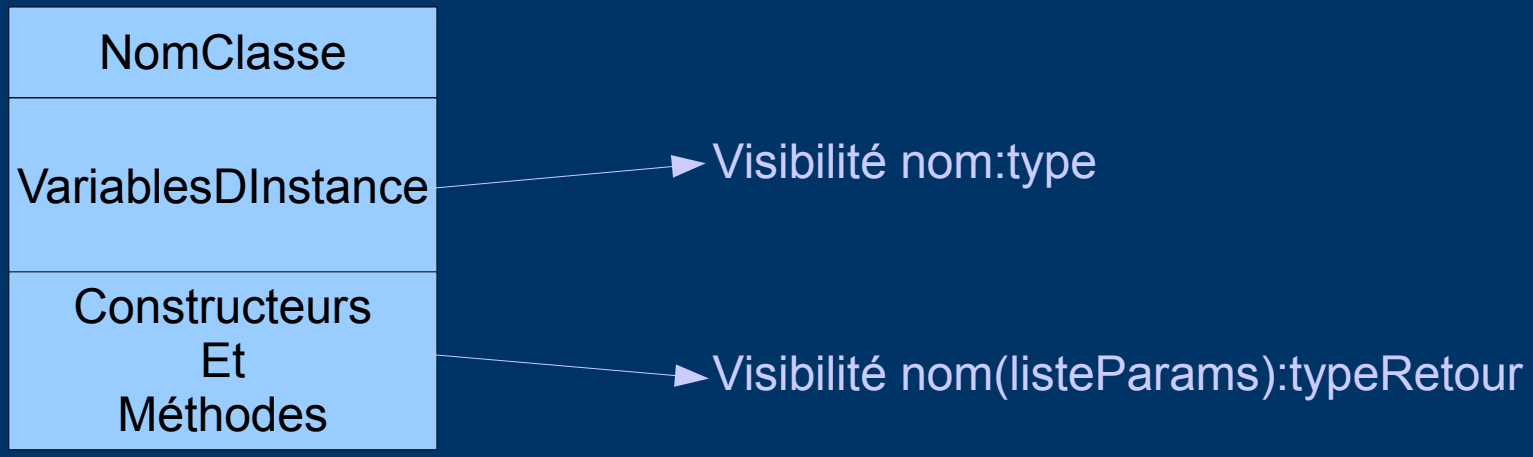
Accessible uniquement depuis la classe de déclaration

Encapsulation des données

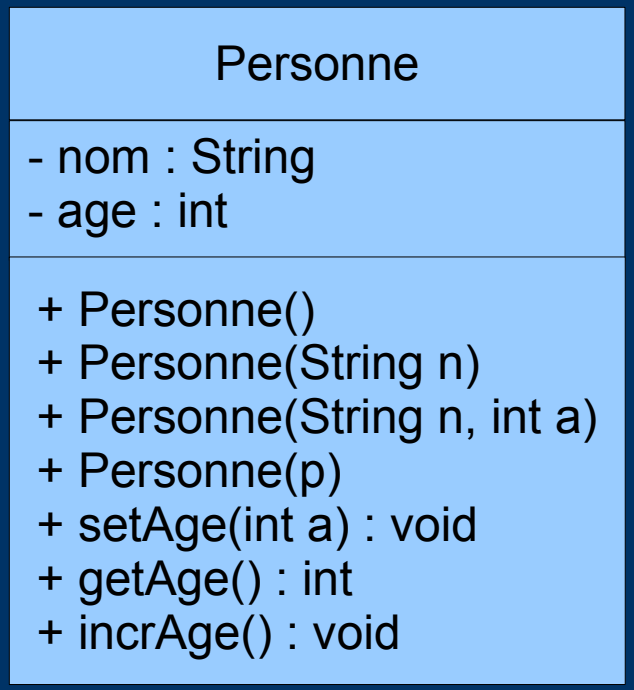
- Principe général
  - Les variables d'instance sont *privées*.
  - Les méthodes et constructeurs d'intérêt général sont *publics*.
  - Les méthodes d'usage interne à la classe sont *privées*.
- Intérêt
  - Les modifications des variables d'instance sont contrôlées par le concepteur de la classe
  - La représentation interne d'une classe peut être changée entre 2 versions sans devoir modifier toutes les classes utilisatrices

Représentation UML simplifiée d'une classe

- UML = Unified Modeling Language
  - Langage permettant d'aider à l'analyse d'un problème et à la conception d'une solution
- Classe en UML



Représentation UML
Exemple de la classe Personne



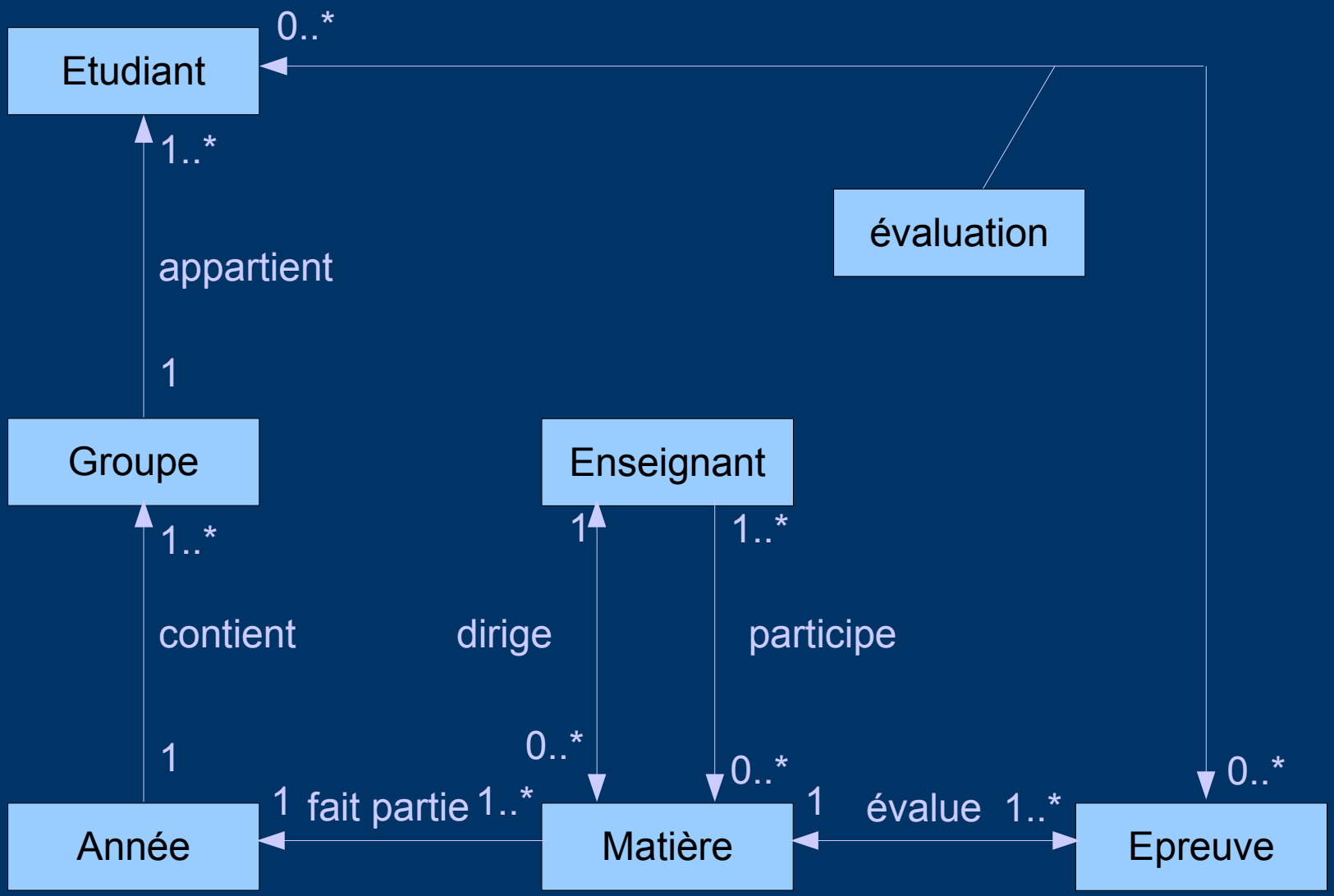

```
Public class Personne {
        private String nom;
        private int age;

        public Personne() {nom="toto";age=20;}
        public Personne(String n) {nom=n;age=30;}
        public Personne(String n, int a) {nom=n;age=a;}
        public Personne(Personne p) {
                nom = new String(p.nom);
                age = p.age;
        }
        public void setAge(int a) {
                if (a >=0) {age = a;}
        }
        public int getAge() {return age;}
        public void incrAge() {age=age+1;}
```

Diagramme de classe (sur un exemple)

- Gestion des notes des étudiants
  - Un étudiant appartient à un groupe
  - Suit certaines matières
  - Il y a un enseignant responsable par matière et plusieurs intervenants
  - Pour chaque matière, les étudiants sont évalués par l'intermédiaire de plusieurs épreuves

Diagramme de classe de l'exemple



Variables d'instance de chacune des classes

- Etudiant
  - Nom:String, prénom:String, numéro:int, demiGroupe:int
- Groupe
  - nom:String
- Année
  - nom:String
- Matière
  - nom:String,coef:float
- Enseignant
  - nom:String,prénom:String
- Epreuve
  - nom:String, type:String,coef:float
- Evaluation
  - nom:float

Code Java initial des différentes classes

```
public class Etudiant {
	private String nom;
	private String prenom;
	private int numEtudiant;
	private int demiGroupe;
}
public class Groupe {
	private String nom;
}
public class Année {
	private String nom;
}
public class Matière {
	private String nom;
	private float coef;
}
```

```
public class Enseignant {
	private String nom;
	private String prenom;
}
public class Epreuve {
	private String nom;
	private String type;
	private float coef;
}
```

Traduction des associations en Java

- Dépend
  - Du couple des cardinalités max de l'association
  - Du sens de navigation privilégié
- Différents cas de cardinalité
  - 1-N
    - Soit une v.i. du type côté 1 dans la classe côté N
    - Soit une collection d'objets de type côté N dans la classe côté 1
    - Soit les deux (uniquement si efficacité critique)
  - N-M
    - Soit une collection d'objets de type côté N dans la classe côté M ou l'inverser ou les deux
    - Soit une nouvelle classe avec 2 v.i. Une de type type côté N, l'autre de type type côté M et une collection des instances de cette classe (solution à privilégier s'il existe des attributs d'association)
  - 1-1
    - Soit fusionner les 2 classes, soit une vi d'un type dans l'autre (si une classe a pour cardinalité min 0, privilégier ce côté-là)

Remarques sur la redondance des infos d'association

- A éviter en général car
  - Perte de place mémoire
  - Risque d'incohérence (on ne modifie qu'une des deux infos synonymes)
- Peut être remplacée
  - En calculant la réciproque de l'info stockée au besoin
- A réserver aux cas où :
  - Efficacité critique
  - Calcul réciproque long (beaucoup de données)
  - Calcul réciproque fréquent
  - Modifications rares/calcul réciproque

Application sur l'exemple

```
public class Etudiant {
        private String nom;
        private String prenom;
        private int numEtudiant;
        private int demiGroupe;}
public class Groupe {
        private String nom;
        private Collection<Etudiant> etudiants;}
public class Année {
        private String nom;
        private Collection<Groupe> groupes;}
public class Matière {
        private String nom;
        private float coef;
        private Année année;
        private Collection<Epreuve> épreuves;
        private Enseignant responsable;}
```

```
public class Enseignant {
        private String nom;
        private String prenom;
        private Collection<Matière> matDirigées;
        private Collection<Matière>
            matDispensées;
}
public class Epreuve {
        private String nom;
        private String type;
        private float coef;
        private Matière matière;
}
public class Evaluation {
        private Etudiant étudiant;
        private Epreuve épreuve;
        private float note;
}
```

Variables et méthodes de classe