Numérique et sciences de l'information

Tous niveaux
Module No 07

Programmation Orientée Objet

Programmation Orientée Objet

- La **Programmation Orientée Objet** consiste à modéliser un ensemble d'éléments d'un domaine du monde réel pour pouvoir le manipuler aisément dans un environnement informatique
- Le résultat de cette modélisation est un groupe d'entités adaptées à un traitement par des logiciels appelés logiciels orientés objet.
- La difficulté de cette modélisation consiste à créer une représentation abstraite et cohérente, sous forme d'*objets*, d'entités ayant une existence réelle –donc qui sont des « objets » au sens classique du terme- (salarié, produit, facture, ...) et d'autres de nature virtuelle (compte, rubrique de paie, ...).

Concept d'objet

- Lors de son apparition, la démarche orientée objet a dérouté les informaticiens qui avaient jusqu'ici soigneusement dissocié les données et les procédures de traitement applicables à ces données.
- Vous avez vu avec Merise, par exemple, des modèles de données conçus totalement indépendamment des modèles de traitement.
- Le concept d'objet regroupe en effet dans un emballage unique les données (propriétés ou attributs) et les traitements (méthodes) qui caractérisent l'entité considérée.

Concept d'objet

- *Simula* a été le premier langage de programmation à implémenter le concept de classes en 1967.
- En 1976, *Smalltalk* implémente les concepts d'encapsulation, d'agrégation, et d'héritage (les principaux concepts de l'approche objet).
- D'autre part, de nombreux langages orientés objets ont été mis au point dans un but universitaire (*Eiffel, Objective C, Loops*, etc.).
- Aujourd'hui tous les développements informatiques sont faits avec des langages orientés objet (*C++, C#, Java, Javascript, Python*..) dont certains sont l'évolution « orientée objet » de langages algorithmiques classiques.

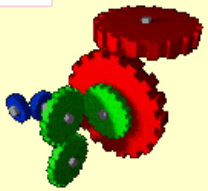
Concept d'objet

- Un **objet** est caractérisé par plusieurs notions.
- Les **attributs** (ou **propriétés**) sont les données caractérisant l'objet.
- Les **méthodes** (ou fonctions membres) caractérisent son comportement, c'est-à-dire l'ensemble des actions (opérations) que l'objet est à même de réaliser.
- Ces opérations permettent de faire réagir l'objet aux sollicitations extérieures (ou d'agir sur les autres objets).
- Remarque : Les méthodes sont étroitement liées aux propriétés, car lers actions entreprises peuvent dépendre des valeurs des attributs, de même qu'elles peuvent les modifier. Cette évidence souligne bien les limites des approches antérieures qui élevaient une cloison étanche entre données (attributs) et traitements (activités)

Concept d'objet

- Un objet (par exemple facture) possède une identité, qui permet de le distinguer des autres objets (client, compte), indépendamment de son état.
- Une **instance** d'objet (la facture émise ce jour pour le client Dupont) possède une identité qui permet de la distinguer des autres instances (autres factures
- De même que pour une entité Merise, on construit généralement cette dernière identité grâce à un **identifiant** (par exemple un produit pourra être repéré par un code, un véhicule par un numéro de série, etc.)
- Instanciation et héritage sont des concepts de base de 'approche objet;

L'objet réconcilie données et traitement

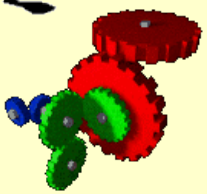


Attributs (modèle)

Méthodes

Classe Individu

Héritage :
Tous les professeurs sont des individus



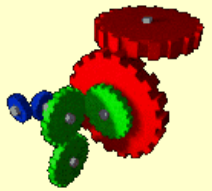
Attributs (modèle)

Méthodes

Classe Professeur

Instanciation : M. Charles M. est un professeur

Professeur.Id = M. Chouette



Attributs (valeurs)

Méthodes

Objet, instance de Professeur

Programmation Orientée Objet

- Unité logicielle autonome qui encapsule données (**attributs** de la classe) et traitements (**méthodes** de la classe).
- Une **classe** est un moule qui permet de fabriquer des structures informatiques qui modélisent le comportement et le fonctionnement des objets de gestion.
- Chaque classe renferme (encapsule) un modèle de données (l'ensemble des attributs) et un modèle de traitements (l'ensemble des méthodes) que l'on peut appliquer sur les données.
- Le sous-ensemble des attributs et méthodes publiques -accessibles au programmeur- constitue l'interface de la classe.

Le concept d'héritage

- Chaque classe hérite des attributs et méthodes de sa classe parente (**héritage** simple) ou de ses différents parents (héritage multiple).
- Un diagramme (graphe d'héritage) permet de factoriser les données et les traitements partagés par plusieurs classes.
- Lorsqu'on parcours un graphe d'héritage depuis la racine jusqu'au classes terminales (feuilles), on passe du général au particulier.
- Les classes terminales sont les classes d'instanciation (production d'objet).
- Les autres classes factorisent la connaissance (classes d'abstraction).

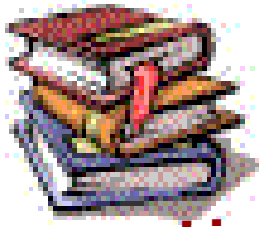
Le concept d'instance

- "**Instances**" de la classe.
- Structure fabriquée à partir du moule.
- Les attributs d'un objet possèdent des valeurs qui peuvent être modifiées grâce aux méthodes associées à la classe dont l'objet est une instance (encapsulation forte).
- Certains langages autorisent la manipulation directe des attributs publics depuis un programme (encapsulation lâche).
- Seule l'encapsulation forte préserve l'indépendance entre objets et garantit l'évolutivité du modèle.

Un exercice pour comprendre l'objet

- Définir une classe d'objets sous forme d'une fonction ayant pour nom l'objet,
- mais dont les instructions permettront, lors de l'appel, d'instancier chacune des propriétés.
- Créer un nouvelle instance de l'objet, c'est appeler la fonction.
- Le mot clef *this* indique que la propriété écrite à sa suite recevra la valeur passée en argument et dont le nom est spécifié à droite du signe égal.

Gestion d'une bibliothèque



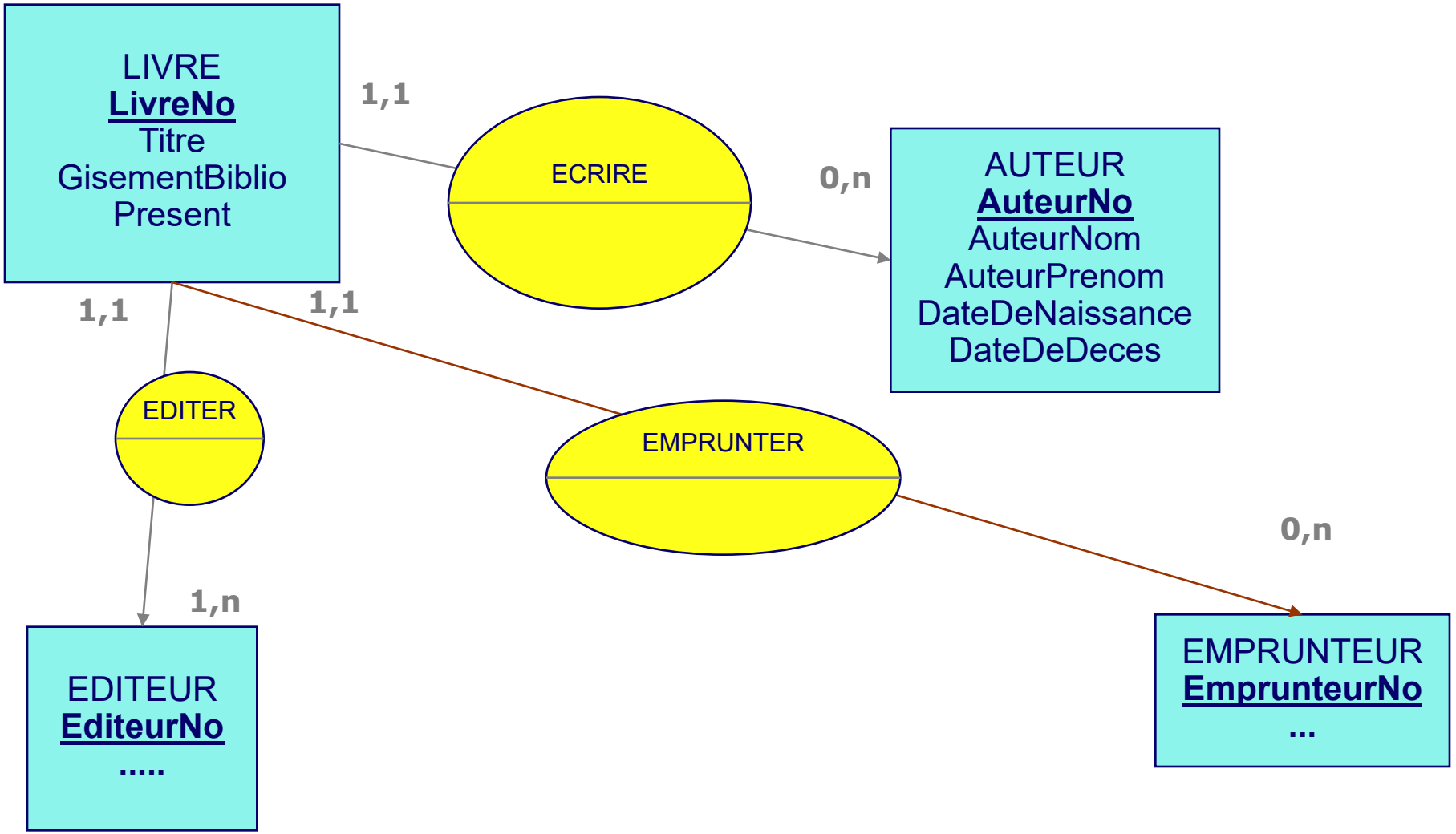
Livre : *Book1 Fables*

Auteur : *Jean de le Fontaine*

Editeur : *Editions Classiques*

Emprunteur : *Edith Dupont*

MCD (Structure des données)



Pour simplifier notre exemple, nous considérerons dans le programme des relations 1,1 - 0,n
UN livre a un auteur et un seul, un éditeur et un seul, un emprunteur et un seul

Définition d'une classe

- Exemple de définition d'une classe : la classe Auteur

```
function Auteur(AuteurNo, AuteurPrenom, AuteurNom, AuteurDateDeNaissance, AuteurDateDeDeces)
{
this.AuteurNo = AuteurNo;
this.AuteurPrenom = AuteurPrenom;
this.AuteurNom = AuteurNom;
this.AuteurDateDeNaissance = AuteurDateDeNaissance;
this.AuteurDateDeDeces = AuteurDateDeDeces;
}
```

Les attributs de la classe

En Merise nous aurions dit les propriétés de l'entité

Pour les curieux : *Chaque langage a ses règles pour définir une classe, instancier un objet, modifier un attribut, définir une méthode, lancer une méthode. Ici notre exemple est inspiré du langage Javascript*

Définition d'une classe

- Définition de la classe Livre

```
function Livre(LivreNo, Auteur, Titre, Editeur, GisementBiblio, Present, EmprunteurNo)
{
this.LivreNo = LivreNo;
this.Auteur = Auteur;
this.Titre = Titre;
this.Editeur = Editeur;
this.GisementBiblio = GisementBiblio;
this.Present = FlagPresent;
this.Emprunteur = Emprunteur;
}
```

Référence à la Classe Auteur

Illustration de l'intérêt fondamental de l'objet : la réutilisabilité
Pas besoin de rappeler tout ce qui concerne la structure de l'auteur

Instanciation d'un objet à partir d'une classe

- Instanciation de l'objet Auteur

- A001= new Auteur ("A001", "Jean", "de La Fontaine", 1621, 1695);
- A002= new Auteur ("A002", "Blaise", "Pascal", 1610, 1655);
- A003= new Auteur ("A003", "Jules", "Vernes", 1800, 1872);
- A004= new Auteur ("A004", "Henri", "Vernes", 1922, "");

Possibilité de modifier a posteriori la valeur d'un attribut :

A001.AuteurPrenom = «Jean-michel »
A004. AuteurDateDeDeces=20XX

Possibilité de récupérer la valeur d'un attribut :
AuteurPrefere = A004.AuteurNom

Affectation de valeurs aux différents attributs de l'objet

Instanciation d'un objet à partir de la classe

- Instanciation de l'objet Livre

- book1 = new Livre ("B001", A001, "Fables", E001, "A01R001P001","P")
- book2 = new Livre ("B002", A001, "Fables", E001, "A01R001P001","P")
- book3 = new Livre ("B003", A002, "Pensées", E001, "A01R002P001","P")
- book4 = new Livre ("B004", A002, "Provinciales", E001, "A01R002P001","P")
- book5 = new Livre ("B005", A003, "5 sem ...", E002, "A01R005P001","P")
- book6 = new Livre ("B006", A003, "20000 lieues ..", E002, "A01R005P001","N", EMP110)
- book7 = new Livre ("B007", A004, "La vallée infer..", E003, "A01R017P001","P")

instance de l'objet Livre

Référence à une instance de l'objet Auteur : l'auteur est Henri Vernes, né en 1922

Affectation de la valeur "La vallée Infernale" à l'attribut Titre de l'instance book7 de l'objet Livre

Concept de méthode

- Possibilité de définir une fonction *Tout(CodeLivre*) listant l'ensemble des informations attachées à un livre.
- L'appel de la méthode *Tout("B001")* permet d'éditer toutes les informations attachées au livre de Jean de La Fontaine.
- On peut aussi définir une fois pour toutes des méthodes comme :
  - *book1.Emprunter(),*
  - *book1. Rendre(), ...*

Que fait un programmeur objet ?

- Imaginons un programmeur objet développant l'interface graphique d'un programme de comptabilité.
- Imaginons qu'il ait définit la **classe "*Fenêtres*".**
- Imaginons qu'il ait définit la **classe "*FenêtreVueCompte*",** héritière de la classe "Fenêtres" et spécialisée dans l'affichage de la situation d'un compte.
- De l'héritage de la classe Fenêtre, cette classe comportera les **attributs *FenêtreVueCompte.Haut*** et ***FenêtreVueCompte.Gauche*** (coordonnées x et y du point supérieur de la fenêtre sur l'écran).
- De la même manière, elle héritera de la **méthode *FenêtreVueCompte.Afficher()*** et de la **méthode *FenêtreVueCompte.Masquer().***
- S'il veut afficher une vue du compte *NoCompte* à la position 100,20 de l'écran, le programmeur écrira :
    - *FenêtreVueCompte.Afficher(NoCompte)*
      FenêtreVueCompte.Haut=20
      FenêtreVueCompte.Gauche=100

Avantage de la POO

- La **POO** (Programmation Orientée Objet) devient modulaire : elle rattache un ou plusieurs blocs de code à chaque objet.
- Ces blocs sont exécutés de manière événementielle.
- Le programmeur n'a pâs à connaître de façon détaillée le fonctionnement des objets qu'il utilise.
- Il suffit qu'il ait une idée de leur fonction et sache comment manipuler les propriétés et les méthodes qui lui sont rattachées.
- Le débogage d'une application se limite à celui du code événementiel.
- Les objets utilisés sont (en principe) totalement débogués ou prêts à l'emploi.