

Module MLBDA
Master Informatique
Spécialité DAC
Cours 2 – Modèles objet et
relationnel-objet

Données complexes

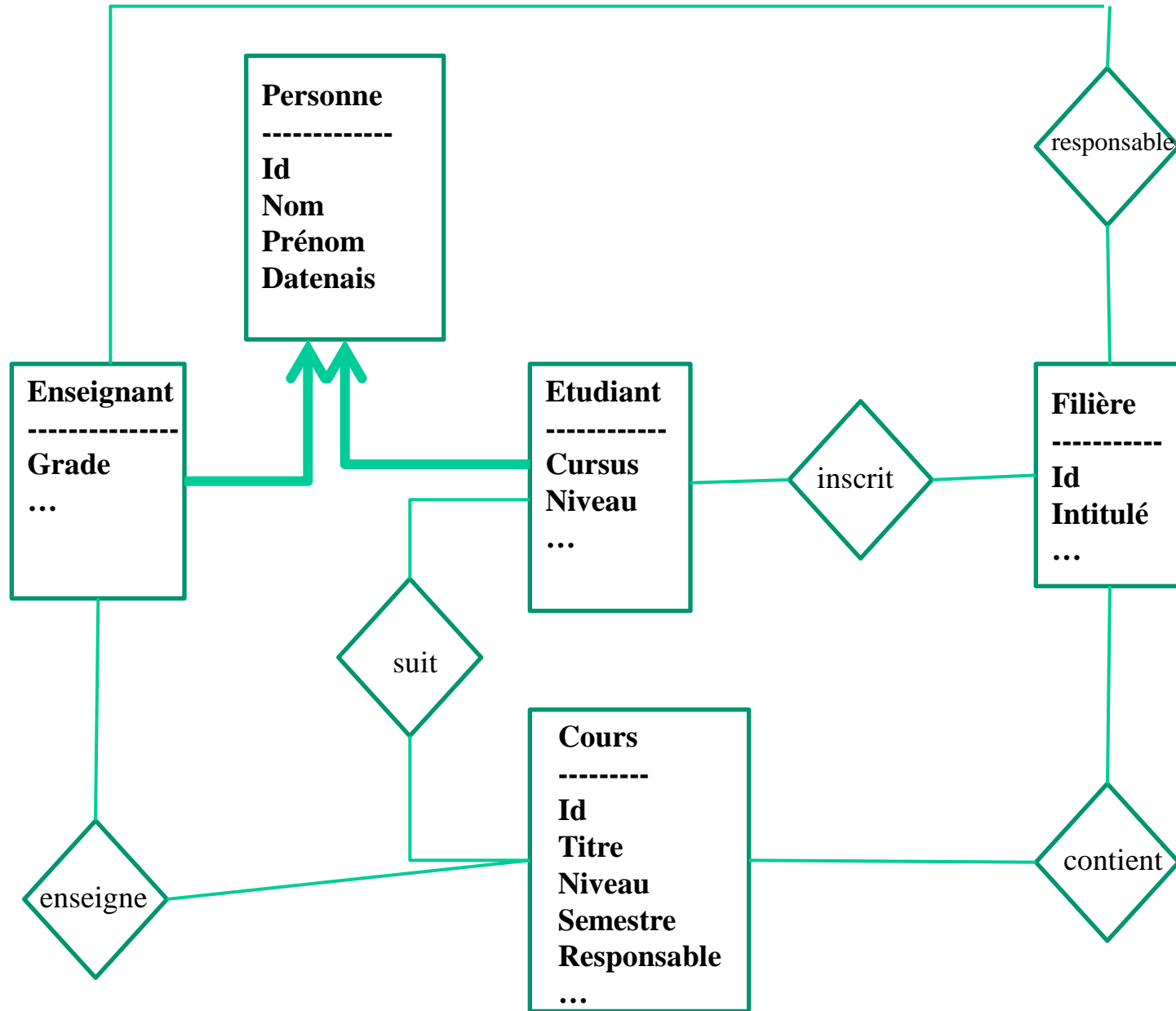


Schéma relationnel

ENSEIGNANT (IdEns, Nom, Prenom, Datnais, Grade)

ETUDIANT (IdEt, Nom, Prenom, Datnais, Cursus, Niveau, Filiere)

COURS (IdC, Titre, Niveau, Semestre)

FILIERE (IdF, Intitule, Responsable)

ENSEIGNE (IdEns, IdC)

SUIT (IdEt, IdC)

CONTENU (IdF, IdC)

RESPONSABLE (IdEns, IdC)

Modèle Objet

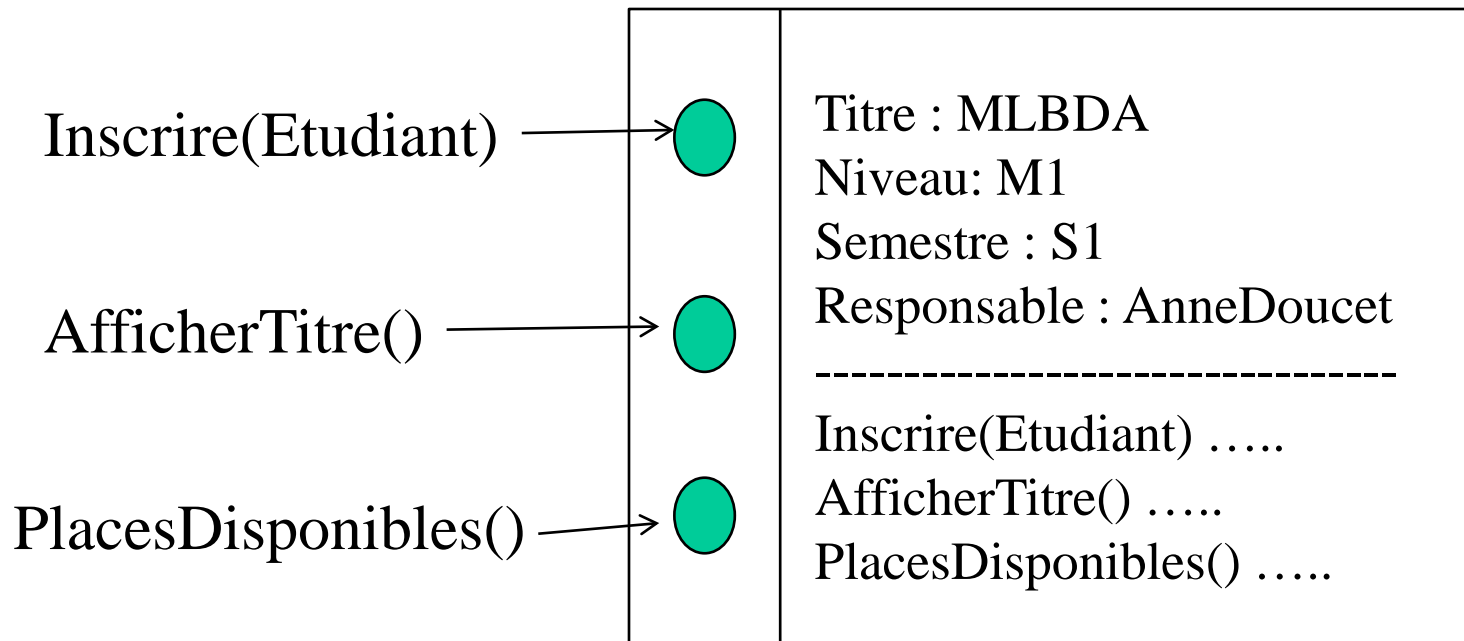
- Principes

- Une entité est représentée par un objet, identifié de façon unique
- Les objets sont reliés entre eux par des liens de composition et des liens d'héritage
- Les opérations sont associées à l'objet

- Concepts :

- Valeur et objet
- Identité d'objet
- Classe
- Héritage

Objet : exemple



Valeur

- Valeur atomique (valeur de type simple : caractère, entier, booléen, ...)
- Valeur complexe : les tuples, les ensembles, les listes, sont des valeurs complexes
 - Ex : [Durand, Paul, 25-10-98, Informatique, {MLBDA, LRC}]

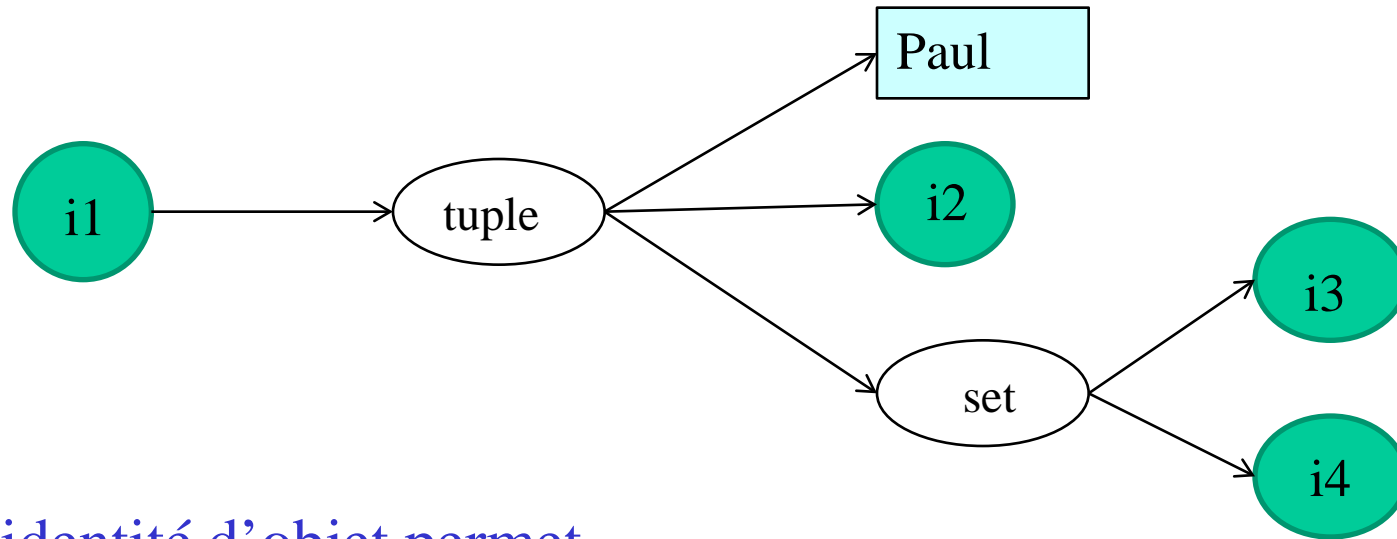
Objet

- Un objet a un identificateur et une valeur.
- Un objet atomique a une valeur atomique
- Un objet complexe est construit en appliquant les constructeurs *tuple*, *set*, *list*, *array*, à des objets atomiques ou complexes.
- L'utilisation des constructeurs est indépendante du type des objets (orthogonalité du système de types)
- Ex : (i1, [MLBDA, M1, S1, i3])
(i2, [Durand, Paul, 25-10-98, Informatique, {i1, i4}])

Identité d'objet

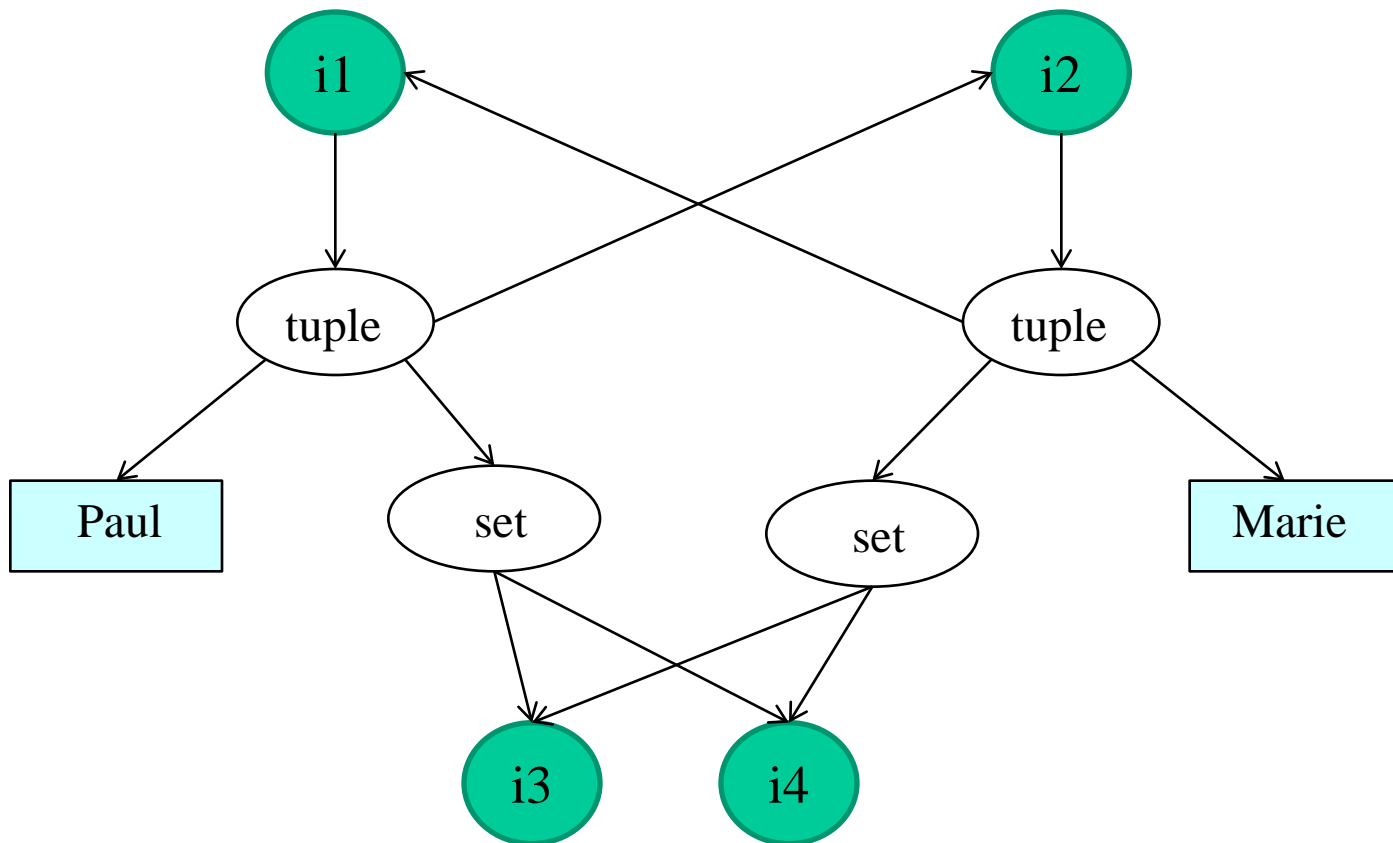
- Chaque objet a une identité indépendante de sa valeur
- L'identificateur est géré par le système (correspond à une clef interne)
- Deux objets sont identiques s'ils ont le même identificateur, et sont égaux s'ils ont la même valeur.
- Les objets peuvent être représentés par un graphe de composition, qui peut comporter des cycles

Identité d'objet

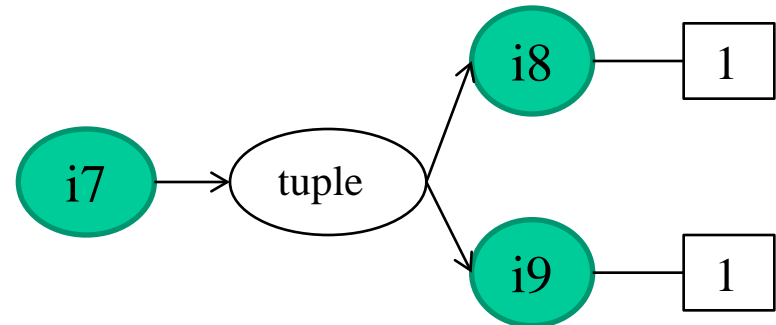
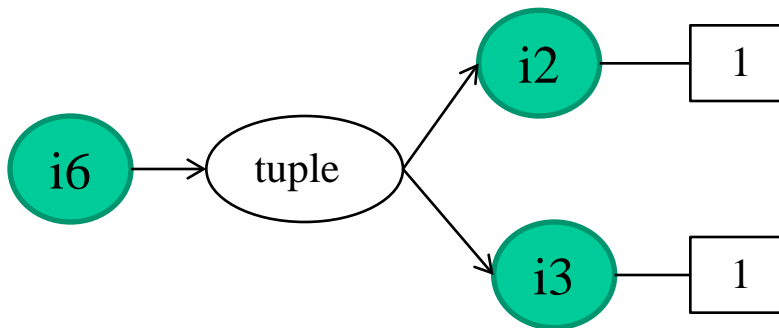
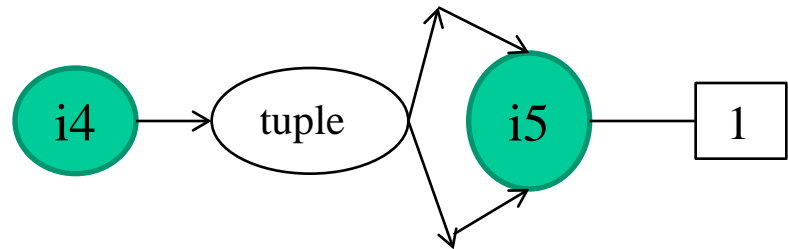
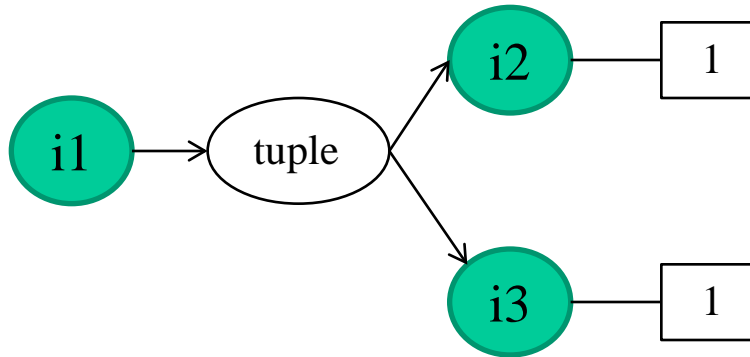


- L'identité d'objet permet
 - le partage d'objets,
 - les cycles entre objets,
 - le maintien automatique des contraintes référentielles,
 - la mise à jour de la valeur sans changer l'identité,
 - plusieurs niveaux de comparaison (égalité profonde, égalité superficielle)

Cycle et partage



Comparaison d'objets



Classe

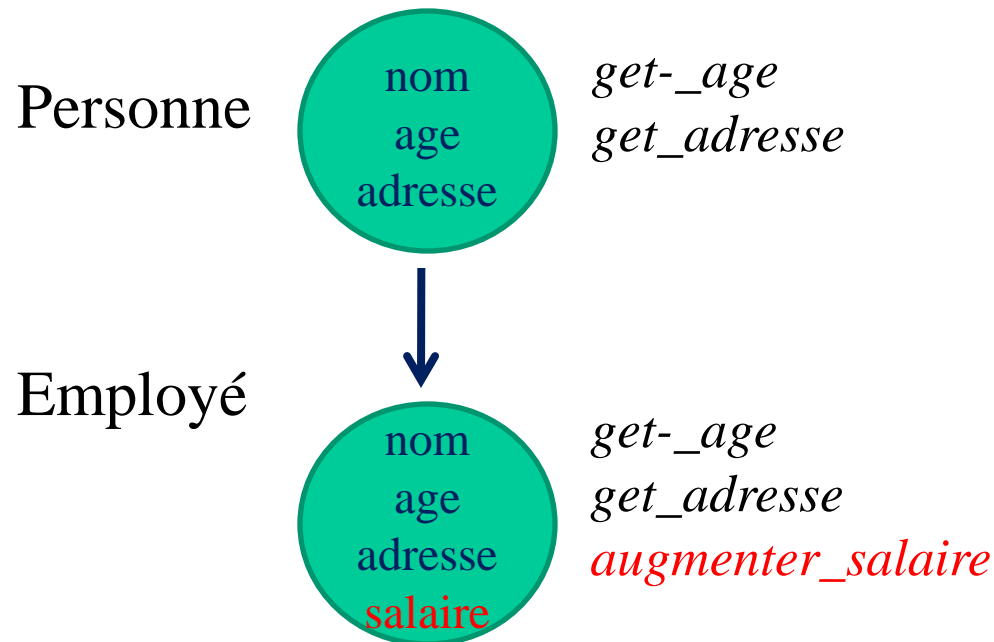
- Les objets partageant des caractéristiques communes (structure et comportement) sont regroupés dans des classes
- Les classes permettent une abstraction des informations et de leur représentation. Ce sont les concepts utilisés pour décrire le schéma de la base.

- Ex:

```
Class Personne
    [nom : string,
    age : integer,
    adresse : string,
    conjoint : Personne,
    enfants : { Personne} ]
method get_age() : integer,
    get_adresse(): string
```

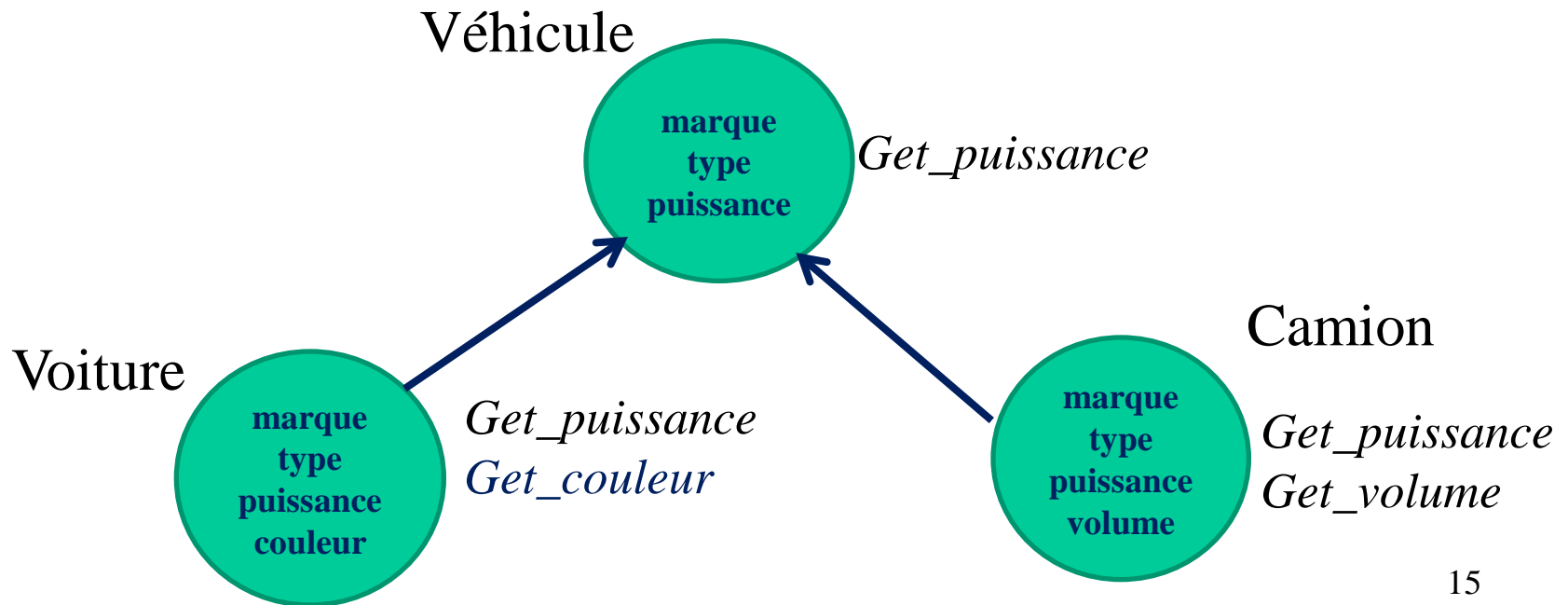
Héritage

- Factoriser des classes ayant des propriétés communes (structure et/ou méthodes). Une sous-classe hérite des propriétés de sa super-classe.
- Spécialisation : affiner une classe en une sous-classe
 - Spécialiser la classe *Personne* en la classe *Employé*



Héritage

- Généralisation : création d'une super-classe regroupant les caractéristiques communes à plusieurs classes.
 - Généraliser les classes *Voiture* et *Camion* en une classe *Véhicule*



Encapsulation

- Un objet est constitué
 - de données
 - d'opérations applicables à ces données
- On distingue l'*interface* (description des opérations applicables à l'objet), de l'*implémentation* (structure des données physique et code des opérations)
- L'utilisateur ne voit que l'interface, les données sont encapsulées.



get_age
get_adresse
modify_adresse

L'apport des modèles objets

- **Identité d'objets**
 - introduction de pointeurs invariants
 - possibilité de chaînage
- **Héritage d'opérations et de structures**
 - facilite la réutilisation des types de données
 - permet l'adaptation à son application
- **Encapsulation des données**
 - possibilité d'isoler les données par des opérations
 - facilite l'évolution des structures de données
- **Possibilité d'opérations abstraites (polymorphisme)**
 - simplifie la vie du développeur

Définitions

BDO = base de données dont les éléments sont des objets (avec identité et opérations)

schéma BDO = graphe de classes des objets de la base

SGBDO =

SGBD : persistance, langage de requêtes, gestion du disque, partage de données, fiabilité des données, sécurité

+

- langages objet (C++, Smalltalk, Java, etc.)
- objets structurés et non structurés
- objets volumineux (multimédias)
- objets complexes (avec associations inter-objets)
- composants et appel d'opérations
- héritage et surcharge d'opération

Persistence

Un objet persistant est un objet stocké dans la base, dont la durée de vie est supérieure au programme qui le crée.

Une donnée est rendue persistante

- par une commande explicite ou
- par un mode automatique :
 - attachement à une racine de persistance (solution LP)
 - attachement à un type de données (solution BD)

Les noms du schéma servent de points d'entrée à la base de données (les relations en relationnel, les noms de classe, ou des objets nommés en BDOO)

ODMG

(Object Database Management Group)

Proposé en 1996, par l'OMG.

Définit les standards pour les BDO

- modèle BDO, extension du modèle OMG
- langage de définition d'objets ODL
- OQL langage de requêtes fonctionnel
- interfaces avec LPO (C++, Smalltalk, Java, etc.)

Interrogation de BD Objet

- **Accès facile à une base objet**
 - via un langage interactif autonome
- **Accès non procédural**
 - optimisations automatiques (ordonnancement, index,...)
 - syntaxe proche de SQL
- **Mises à jour limitées via les méthodes**

OQL : Langage de requêtes fonctionnel

Applications des SGBD objet

- Domaines d'applications
 - Internet/Intranet
 - technique
 - gestion de réseau, CAD, CASE, CAM, etc.
 - systèmes bancaires et financiers
 - systèmes d'information géographiques
 - systèmes d'information médicaux
- Besoins
 - objets structurés et non structurés
 - associations complexes
 - sources de données hétérogènes

Bilan

- Modélisation simple de données complexes
 - Bonne intégration avec les langages objet
 - Langage de requêtes fonctionnel
-
- Nombreux systèmes dans les années 89-90
 - Forte concurrence du marché avec les SGBDR

L'objet-relationnel

- **Relationnel** : tables, attributs, domaine, clé

+

- **Objet** : collections, identifiant d'objet, héritage, méthodes, types utilisateurs, polymorphisme
- Pas de classe : type abstrait de données avec méthodes, et table
- Persistance via les tables

L'objet-relationnel

- Extension du modèle relationnel
 - attributs multivalués : structure, liste, tableau, ensemble, ...
 - héritage sur relations et types
 - domaine type abstrait de données (structure cachée + méthodes)
 - identité d'objets
- Extension de SQL
 - définition des types complexes avec héritage
 - appels de méthodes en résultat et qualification
 - imbrication des appels de méthodes
 - surcharge d'opérateurs

Exemple objet Personne

NSécu **1234**

Nom **Paul**

Adresse **Paris**

Enfants **Prénom : Léa** **Prénom : Max**
 Age : 7 **Age : 5**



Voitures **Marque : Ferrari**
 Type : F355
 Photo :



Marque : Citroën
Type : 2CV
Photo :



Exemple de table et objet

NSécu	Nom	Adresse	Enfants		Voitures		
1234	Paul	Paris	Prénom	Age	Marque	Type	Photo
			Léa	7	Ferrari	F355	
			Max	5			
					Citroën	2CV	

Objet Personne

Les concepts

- Extensibilité des types de données
 - Définition de types abstraits
 - Identité d'objet
- Support d'objets complexes
 - Constructeurs de types (tuples, set, list, ...)
 - Utilisation de référence (OID)
- Héritage
 - Définition de sous-types
 - Définition de sous-tables

Le modèle SQL3 (ANSI99)

Extension objet du relationnel, inspirée de C++

- **type** = type abstrait de données avec fonction
- **objet** = instance de type référencée par un OID (défaut)
- **collection** = constructeur de type
 - **table, tuple, set, list, bag, array**
- **fonction**
 - associée à une base, une table ou un type
 - écrite en SQL, SQL3 PSM (Persistent Stored Module) ou langage externe (C, C++, Java, etc.)
- **sous-typage et héritage**
- **association** = contrainte d'intégrité inter-classe

Types atomiques (SQL3 Oracle)

- Types atomiques
 - `Varchar2(<longueur>)`
 - `Number(<longueur>)`
 - `Date`

Ex :

```
create type nsecu varchar2(15);  
create type taille number(3) ;  
create type datenais Date;
```

Types objet (SQL3 Oracle)

Types objet

```
Create type <nom-type> as Object (  
    (<nom-attribut> [ref] <type>, )+  
    (<declaration-methodes> ,)*  
);
```

Ex :

```
create type personne as Object (  
    nom Varchar2(10),  
    nss nsecu,    % type défini par l'utilisateur  
    datenais Date) ;
```

Types ensemblistes (SQL3 Oracle)

Types ensemblistes :

- Table (ensemble avec doublons)
- Varray (collection ordonnée avec doublons)

```
Create type <nom-type> as  
    (Table | Varray(<longueur>)) of <type>;
```

```
create type retraits as Table of personne;  
create type centenaires as Varray (50) of  
    personne ;
```

Références (type REF)

LesEtudiants

NOM	COURS		
Max	MLBDA	Doucet	20
Lucie	MLBDA	Doucet	20
Marie	MLBDA	Doucet	20
Paul	LRC	Ganascia	20

LesCours

TITRE	RESP	HEURES
MLBDA	Doucet	20
LRC	Ganascia	20

```
Create type Cours as object (  
  Titre varchar2(15),  
  Resp varchar2(20),  
  Heures number(2) );
```

```
Create type Etudiant as object (  
  nom varchar2(15),  
  suit Cours);
```


Références (type REF)

Le type **REF** est un pointeur logique sur un objet. Il permet de référencer un objet par son OID (permet le partage d'objets). L'objet peut être consulté, modifié.

LesEtudiants

NOM	COURS
Max	
Lucie	
Marie	
Paul	

LesCours

TITRE	RESP	HEURES
MLBDA	Doucet	20
LRC	Ganascia	20

```
Create type Cours as object (  
  Titre varchar2(15),  
  Resp varchar2(20),  
  Heures number(2) );
```

```
Create type Etudiant as object (  
  nom varchar2(15),  
  suit REF Cours); /*Le champ suit fait référence à un objet de type Cours.
```

Utilisation des REF

- On utilise le type **REF** pour faire référence à un objet. Les associations se modélisent à l'aide d'attributs en utilisant le type REF.
- Association 1-1 :

```
Create type personne as object  
  (nom Varchar2(10),  
   conjoint ref personne,...) ;
```

Une personne a un conjoint, qui est une personne (qui a un conjoint)

Association 1: N

- Le type de l'attribut doit être une collection (table ou varray)

```
Create type personne;
```

```
Create type ens-enfant as table of ref  
personne;
```

```
Create type personne as object  
(nom varchar(10),  
père ref personne,  
enfants ens-enfant, ...);
```

Association N:M

- Le type des attributs doit être une collection

```
Create type Cours; /* type incomplet
Create type ens-cours as table of ref Cours;
Create type etudiant as object (
    nom varchar2(15),
    suit ens-cours);

Create type ens-etud as table of ref etudiant;
Create type Cours as object (
    titre varchar2(15),
    resp ref Personne,
    suiviPar ens-etud);
```

Méthodes (déclaration)

- Fonctions ou procédures associées à un type d'objet.
- Modélisent le comportement d'un objet
- Ecrites en PL/SQL ou JAVA, et sont stockées dans la base.

```
Member function <nom-fonction>
```

```
    [ (<nom-para> in <type>, ...) ] return  
    <type-resultat>
```

```
Member procedure <nom-proc>
```

```
    [ (<nom-para> in <type>, ...) ]
```

Méthodes (implémentation)

Le corps de la méthode est défini dans la classe du receveur.

```
Create type body <type-objet> as  
    <declaration-methode> is  
    <declaration var-locales>  
begin  
    <corps de la methode>  
end;
```

Exemple

```
create type personne as Object (  
    nom Varchar2(10),  
    nss nsecu,  
    datenais Date,  
    member function age return Number) ;  
  
create type body personne as  
    member function age return Number is  
begin  
    return sysdate - datenais;  
end;
```

Sous-typage et héritage

```
Create type jour-ouvré as varchar2 (15);  
create type personne as object  
    (nom varchar2(10), adresse varchar2(30), datenais  
    date) not final;  
  
create type salarie under personne  
    (affectation varchar2(10), repos jour-ouvré);  
  
create type étudiant under personne  
    (université varchar2(20), no-étudiant number(10)) ;  
  
create table LesPersonnes of personne  
    (primary key (nom)) ;  
  
create table LesSalariés under LesPersonnes of  
    salarié ;
```


Stockage des données

Les objets sont stockés dans des relations ([table](#)),

- comme n-uplet,

```
Create table <nom-table> of <nom-type>;
```

- ou comme attribut d'un n-uplet.

```
Create table <nom-table> (  
    (<nom-attribut> [ref] <type>, )+  
);
```

Exemple

Stockage d'objet comme n-uplet :

```
Create table LesPersonnes of personne;
```

La table `LesPersonnes` stocke les objets de type `personne` (une instance par n-uplet).

Stockage d'objet comme attribut d'un n-uplet :

```
Create table LesFamilles (  
    nom Varchar2(10),  
    pere personne,  
    mere personne);
```

Le champ `père` (`mere`) de la relation `LesFamilles` stocke un objet de type `personne` (l'objet est stocké comme attribut d'un n-uplet).

Stockage des collections

```
Create type Toeuvre as object (  
    titre varchar2(30),  
    type varchar2(20));  
Create type ens_oeuvres as table of Toeuvre;  
Create table Artiste (  
    nom varchar2(30),  
    Datenais Date,  
    œuvres ens_oeuvres)  
Nested table œuvres store as Tab1;
```

Artiste			Tab1			
Nom	Datenais	œuvres	NTindex	oeuvre	type	Date
Picasso	1881	index1	index1	Guernica	Huile	1937
			index1	Le rêve	Huile	1932
Van Gogh	1853	index2	index2	Iris	Huile	1889
			index1	Tête de femme	Sculpture	1931
			index2	tournesols	Huile	1888

Stockage des collections

On définit une table imbriquée (**nested table**) pour les attributs ensemblistes :

```
Create table <nom-table> of <nom-type>  
  nested table <nom-attribut> store as <nom-table-  
  imbriquée>;
```

Remarques :

- On déclare des 'nested table' pour le type ensembliste table uniquement;
- Pas de 'nested table' pour le type ensembliste Varray.
- On déclare une 'nested table' pour chaque champ ensembliste de la table.
- On déclare une 'nested table' pour chaque champ ensembliste d'une table imbriquée.

Example

```
Create type A as table of varchar2(20);
```

```
Create type B as Object (  
    att1 varchar2(10), att2 number);
```

```
Create type EnsB as table of B;
```

```
Create type C as object (  
    att3 varchar2(20), att4 EnsB);
```

```
Create type D as object (  
    att5 C, att6 A );
```

```
Create table LesD of D
```

```
Nested table att5.att4 store as tab-att4, nested  
table att6 store as tab-att6;
```

Example

Create type **A** as table of varchar2(20);

Create type B as object (

att1 number, att2 **A**);

Create type **EnsB** as table of B;

Create type C as object (

att3 varchar2(10), att4 **EnsB**);

Create table LesC of C nested table att4
store as tab-4 (nested table att2 store
as tab-2);