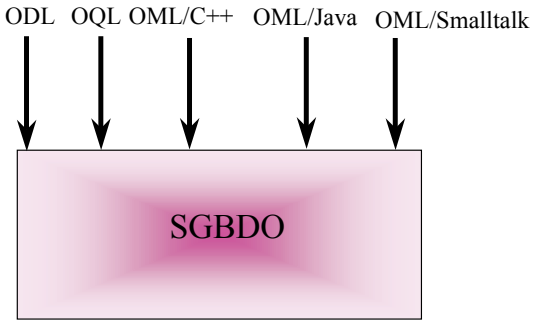
L 'ODMG

- **Proposition d 'un "standard" pour les SGBDOO**
  - *Object Database Management Group* :
    - Fondé en septembre 91 par 5 constructeurs:
      - O2 Technology, Objectivity, Object Design, Ontos, Versant
    - Version 1.0 publiée en 1993
    - Version 2.0 en 1996 avec: 10 auteurs
      - POET Soft, Lucent , Barry &Ass., Amer Man. Syst., Windward Sol.,
  - *est renommé Object Data Management Group* en 1998
    - Sun rejoint l 'ODMG pour travailler sur le binding Java
    - *Version 3.0 en Janvier 2001 ODMG Java Binding (donnera naissance Java Objects ou JDO) et dissolution de l'ODMG*

- **Travaux de l'ODMG :**
  - définition d'un modèle objet de référence ODL
  - définition d'un langage déclaratif OQL
  - définition des différents binding avec C++, Smalltalk et Java

4.2. PROPOSITION DE L 'ODMG

- **Adaptation du modèle objet de l'OMG**
- **Interfaces d'accès à un SGBDO**



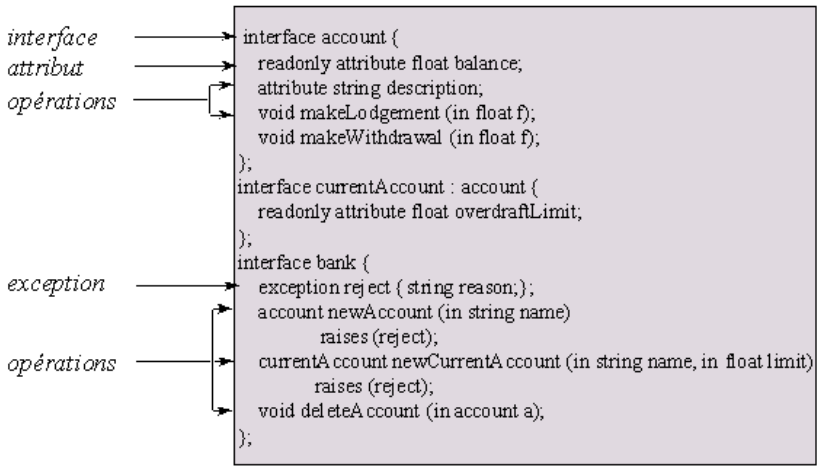
PLAN

- **ODL**
  - extension de IDL, meta-schema
- **OML binding**
  - OML/C++ binding
  - OML/Java binding
- **OQL**

1. ODL

- **Object Definition Language :**
  - extension du langage IDL défini par l 'OMG
    - IDL :
      - langage pour décrire les interfaces des objets
      - langage pivot entre applications
      - pour générer des squelettes de programme dans les langages de programmation des applications
  - les BD objets nécessitent des adaptations/extensions :
    - instances de classes, collections, associations, persistance, transactions

1. LE LANGAGE IDL



1. ODL

- **Syntaxe :**

interface classname {

extent	extname;
keys	keyatt [, keyatt...];
attribute	attType attname; ...
relationship	relType pathname **inverse** class::invpath;
operations	….}

- **Exemple :**

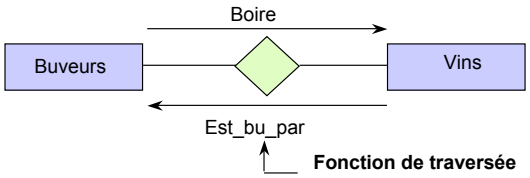
Interface PART {

extent	parts;
keys	part_id;
attribute	String part_id;
relationship	Set<Manufac> manufacture
	inverse Manufac::mader; }

1. LES ATTRIBUTS

- **Propriétés permettant de mémoriser :**
  - un littéral ou un objet
  - possède un nom et un type de ses valeurs légales
- **Sont vus à l 'aide de deux fonctions :**
  - Set_value
  - Get_value
- **Un attribut n'est pas forcément implémenté.**

1. LES ASSOCIATIONS



- **Associations binaires, bi-directionnelles de cardinalité (1:1), (1:N), (N:M).**

```
interface Buveurs { ...
   Relationship LIST<Vins> boire inverse Vins::est_bu_par;
}
interface Vins { ...
   Relationship SET<Buveurs> est_bu_par inverse Buveurs::boire;
}
```

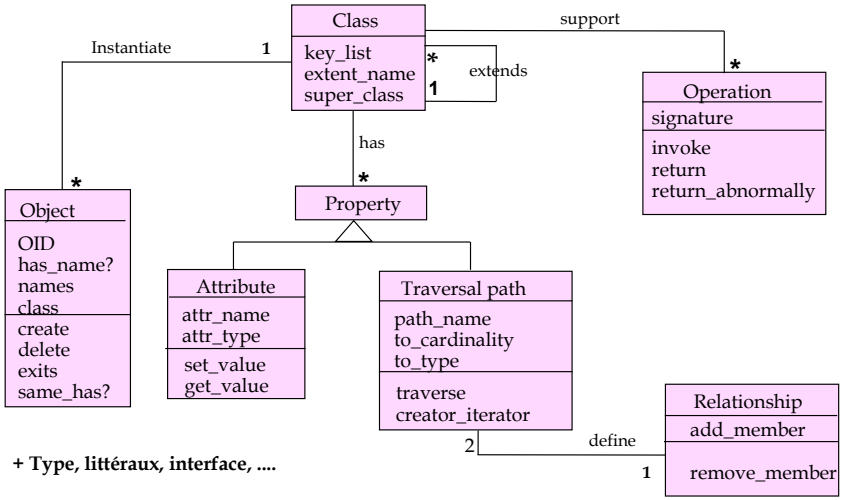
1. LES ASSOCIATIONS ...

- **Opérations:**
  - Add_member, Remove_member,
  - Traverse, Create_iterator_for

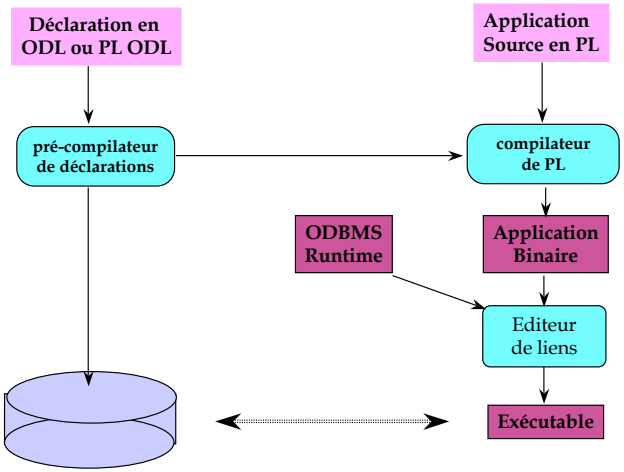
- **Le SGBDO est responsable du maintien de l'intégrité :**
  - Référence dans les deux sens si inverse déclaré
  - Ce n'est pas le cas pour un attribut valué par un objet : pas de chemin inverse, pas d'intégrité référentielle

```
interface Buveurs { ...
    attribute LIST<Vins> boire ;
}
```


1. LE META-MODELE DE L 'ODMG

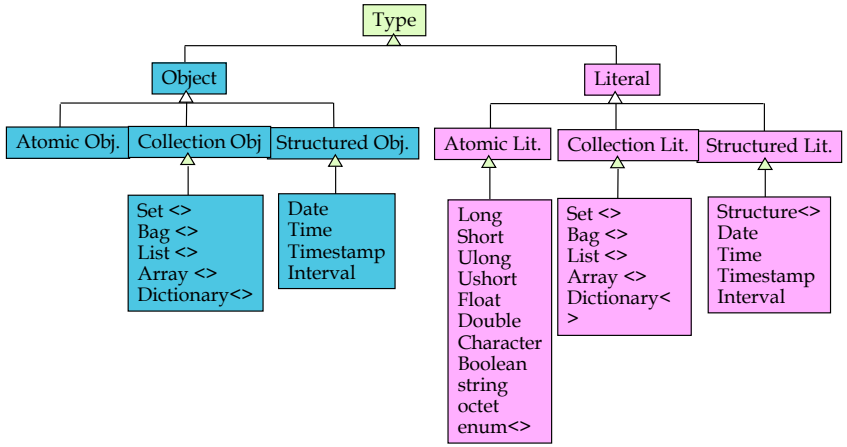


1. ARCHITECTURE



2. OML Binding

- **Il s 'appuie sur un modèle Objet-Valeur**



2. OML Binding

- **Composer d ’un ensemble de classes ou package pour la gestion :**
  - de la persistance
  - la manipulation des collections
  - des transactions :
    - begin, commit, abort, checkpoint
  - des accès aux BD :
    - open, close, lookup,...

2. OML/C++ Binding

- **Modèle de persistance par héritage :**
  - toute classe pour obtenir des instances persistantes doit hériter de la classe Pobject

    Person * p = new persistent Person("toto ");
  - utilisation d ’un pointeur spécial PREF pour référencer les instances d ’une classe persistante

    REF<T> : pointeur générique
- **Exemple :**

```
Class Personne : public Persistent_Object {
    public :
        int age;
        REF<Personne> pere;
        SET<REF<Personne>> amis;
}
```


2. OML/C++ : EXEMPLE DE SESSION

```
void main () {
    database *db = Database::open(« MaBD »);
    transaction t* = new Transaction();
    REF<Personne> toto;
    t->begin();
    toto=db->lookup_object ("toto");
    toto->pere.Afficher();
    toto->pere.Augmenter(10);
    t->commit();
    db->close();
}
```


2. OML/Java Binding

- **Modèle de persistance par atteignabilité**
  - nommage des objets :
    - fournit un point d ’entrée pour la navigation des objets dans un graphe
    - manipulation transparente
  - chaque nom est unique à l ’intérieur d ’une BD
    - technique à réserver pour un petit nombre d ’objets
- **Les méthodes :**
  - *public synchronized void **bind**(Object obj, String name)*
  - *public synchronized Object **lookup**(String name) throws ObjectNameNotFoundException*
  - *public synchronized void **unbind**(String name) throws ObjectNameNotFoundException*

2. ODMG Java Binding

- ***public synchronized void** bind**(Object obj, String name)***
  - **Parameters**:
    - obj - the object being bound to a name in the database.
    - name - the name used to bind an object to in the database.
  - **Throws**:
    - ObjectNameNotUniqueRuntimeException
      - a run-time exception if there is an object already bound to this name in this database.
    - DatabaseClosedRuntimeException
      - a run-time exception if the database has already been closed.
    - IllegalArgumentException
      - a run-time exception, if the name is invalid
    - TransactionNotInProgressException
      - a run-time exception, if there is no transaction identified as being associated with this database.

2. ODMG Java Binding

- ***public synchronized Object** lookup**(String name)***
  throws ObjectNameNotFoundException
  - **Parameters**:
    - name - the name used to bind an object to in the database.
  - **Throws**:
    - ObjectNameNotFoundException
      - if there is no such named object in this database.
    - DatabaseClosedRuntimeException
      - a run-time exception if the database has already been closed.
    - IllegalArgumentException
      - a run-time exception if the name is invalid
    - TransactionNotInProgressException
      - a run-time exception if there is no transaction identified as being associated with this database.

2. ODMG java binding

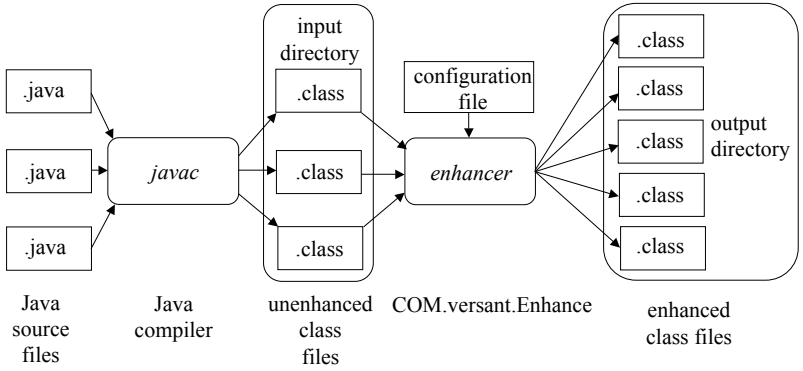
- ***public synchronized void** unbind**(String name)***
  throws ObjectNameNotFoundException
  - **Parameters**:
    - name - the name used to bind an object to in the database.
  - **Throws**:
    - ObjectNameNotFoundException
      - if there is no such named object in this database.
    - DatabaseClosedRuntimeException
      - a run-time exception if the database has already been closed.
    - TransactionNotInProgressException
      - a run-time exception, if there is no transaction identified as being associated with this database.

VERSANT : CreatePerson.java

```
CreatePerson.java
// Transparency is achieved  by a post-processor, //com.versant.Enhance
import  com.versant.trans.*;
import  com.versant.fund.*;
import  java.util.*;

public class CreatePerson
{
  public static void main (String[]  args)
  {
    if (args.length !=3)  {
       System.out.println
          ("Usage: java  CreatePerson <database> <name>  <age> );
          System.exit(1);
    }
    String database = agrs[0];
    String name     = agrs[1];
    int   age       = Integer.parseInt ( args[2] );

    Properties prop = new Properties();
    prop.put("database", database  );
    Capability capability =  new  NewSessionCapability ();
    TransSession  session = new TransSession(prop, capability);
    //  class Person is in the same directory
    Person person = new Person(name, age);
    session.endSession();


  }//end main

}//class  end
```

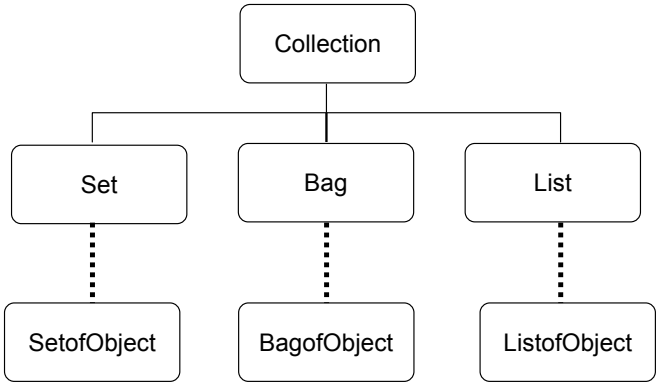
VERSANT
THE DATABASE FOR OBJECTS®

VERSANT : Enhance the Java Classes



VERSANT

2. LES COLLECTIONS ODMG



2. L'INTERFACE COLLECTION

Interface Collection

Object	add(Object object)
boolean	contains(Object object)
Enumeration	elements()
boolean	existsElement(String predicate)
boolean	isEmpty()
Collection	query(String predicate)
int	remove(Object object)
Enumeration	select(String predicate)
Object	selectElement(String predicate)
Int	size()

2. LES INTERFACES BAG ET LIST

interface Bag extends Collection

Bag	difference()
Bag	intersection(Bag bag)
int	occurrences(Object object)
Object	put(Object object)
Bag	union(Bag bag)

interface List extends Collection

void	add(int index, Object obj)
void	append(List list)
List	concat(List list)
Object	get(int index)
void	put(int index,Object object)
Object	remove(int index)
void	reverse()
void	unique()

3. SQL OBJET : FONCTIONNALITES

- **La description de données :**
  - schéma + vues + contraintes d'intégrité
  - il faut utiliser le langage unique de programmation
- **La manipulation des données :**
  - en général pas de commande INSERT, UPDATE ou DELETE
    - les opérations de MAJ sont gérés via OML
  - supporte le SELECT
- **Les droits :**
  - ils sont gérés par Unix

3. GENERALITES

- **OQL :**
  - issu du langage O2SQL développé à l 'INRIA
    - langage fonctionnel intégrant toutes les fonctionnalités du SQL standard (syntaxe proche de SQL 92)
  - doit permettre des optimisations automatiques
    - ordonnancement des opérations, index,…
  - Reste conforme au modèle de l'ODMG :
    - application d'opérateurs aux collections : extensions ou imbriquées
    - permettre de créer des résultats littéraux, objets, collections, …
  - Supporte des mises à jour limitées via les méthodes
    - attention danger!…
  - Brise le principe d 'encapsulation

3. CONCEPTS NOUVEAUX

- **Expression de chemin mono-valuée**
  - Séquence d'attributs ou associations mono-valués de la forme X1.X2…Xn telle que chaque Xi à l'exception du dernier contient une référence à un objet ou un littéral unique sur lequel le suivant s'applique.
  - Utilisable en place d'un attribut SQL
- **Collection dépendante**
  - Collection obtenue à partir d'un objet, soit parce qu'elle est imbriquée dans l'objet ou pointée par l'objet.
  - Utilisable dans le FROM

3. FORME DES REQUETES

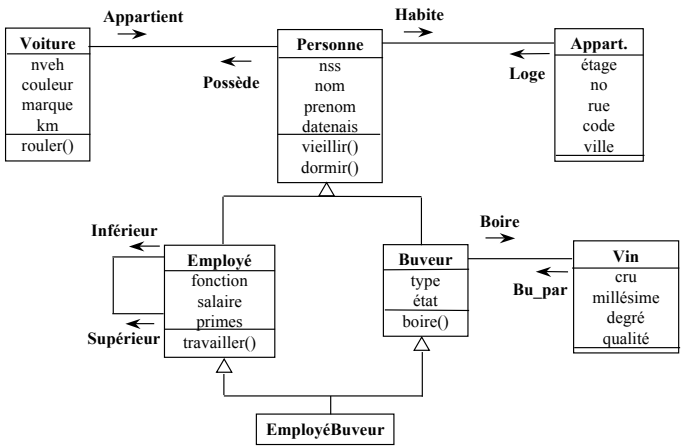
- **Forme générale d'une requête**
  - Expression fonctionnelle mixée avec un bloc select étendu

    Select [<type résultat>] (<expression> [, <expression>] ...)

    From x in <collection> [, y in <collection>]...

    Where <formule>
- **Type résultat**
  - automatiquement inféré par le SGBD
  - toute collection est possible (bag par défaut)
  - il est possible de créer de nouveaux objets en résultat d 'une requête
- **Syntaxe très libre, fort contrôle de type**

3. EXEMPLE DE REFERENCE



3. REQUETE SIMPLE

- **Calcul d'une expression**
  > ((string) 10*5/2) || "toto"
- **Accéder un attribut d'un objet nommé**
  > mavoiture.couleur
- **Appeler une méthode**
  > mavoiture.rouler();
- **Expression de chemin**
  > mavoiture.appartient.nom;

3. PARCOURS D 'ASSOCIATION

- **Parcours d 'association monovaluée :**
  - *avec sélection de structure (défaut)*

  > select struct (name: b.nom, city: b.habite.ville)
  from b in buveurs
  where b.type = 'gros ';
  ===> littéral bag <struct<name:string,city:string>)>

- **Parcours d 'association multi-valuée :**
  - *Utilisation de collections dépendantes*

  > select b.nom, b.prenom
  from b in buveurs, v in b.boire
  where v.cru = "volnay » ;
  ==> litteral bag<struct<nom:string,prenom:string>>

3. RESULTATS IMBRIQUES

- **Imbrication des select au niveau du select**

  > select distinct struct (nom : e.nom,
  inf_mieux_payes :
  select i
  from i in e.inferieur
  where i.salaire > e.salaire))
  from e in employes ;
  ===> litteral set <struct <nom: string,
  inf_mieux_payes : bag <employes>>

et aussi au niveau du FROM (requête collection)

3. CALCULER UNE JOINTURE

- **Forme classique :**

```
select  v.nveh, p.nss
from v in Voitures, p in Personnes
where v.marque = p.nom ;
```

- **Imbrication OQL :**

```
select struct ( nveh : v.nveh, nss : select p.nss
                                     from p in Personnes
                                     where (v.marque=p.nom))
from v in Voitures;
```


3. REQUETES QUANTIFIEES

- **Quantificateur universel**
  - *for all x in collection : predicat(x)*

```
for all b in Buveurs : b.age < 18
```

- **Quantificateur existentiel**
  - *exists x in collection: predicat(x)*

```
exists v  in  Employés.possède : v.marque = "Renault"
```


3. CALCUL DES AGREGATS

- **Similaire à SQL mais avec possibilité de prédicats :**

```
select e
from e in Employes
group by (bas : e.salaire < 7000,
      moyen : e.salaire >= 7000 and e.salaire < 21000,
      haut : e.salaire >= 21000)
```

===>struct<bas: set<emp>, moyen:set<emp>, haut:set<emp>>

3. INVOCATION DE METHODES

- **En résultat ou dans le critère :**

```
> select distinct e.nom, e.habite.ville, e.age()
  from e in Employes
  where e.salaire > 10000 and e.age() < 30
  ===> litteral de type set <struct>
```

3. CREATION D 'OBJETS

- **Expressions de constructions**
  - Si C est le nom d'une classe, p1, ..., pn des propriétés de la classe et e1, ..., en des expressions
  - alors C(p1 : e1, ..., pn : en) est une expression de construction

- **Création d'objet**

  employe (nss:15603300036029, nom:"jean", salaire: 100000)

  employe (select struct(nss:b.nss, nom:b.nom, salaire: 4000)
  from b in buveurs
  where not exist e in employes : e.nss=b.nss)

3. Définition d'objets nommés via Requêtes

- **Définition d 'objets nommés :**
  - *Define <name> as <query>*
  - permet de définir un objet de nom <name> calculé par la requête OQL <query>

- **Exemple:**

  Define Cesars as
  Select b
  From b in Buveurs
  Where b.prenom = "Jules"

3. EXPRESSION SUR LES COLLECTIONS

- **Conversion de collections :**

  element (select v.marque
  from v in voitures
  where v.numero = "120 abc 75")

  ===> string

- **Aplatir des collections :**

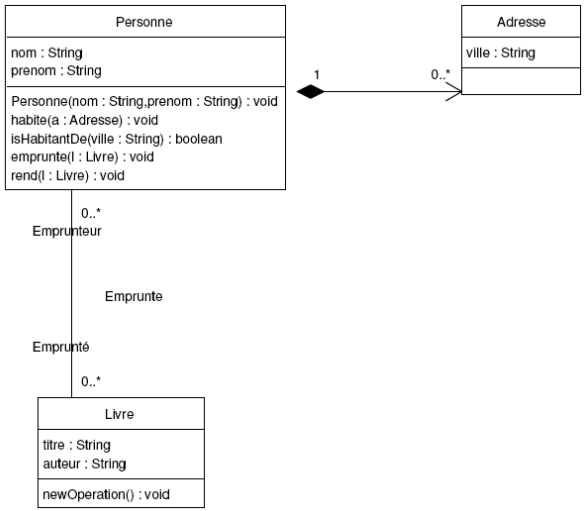
  flatten (select b.nom, select v.millesime
  from v in b.boire
  where v.cru = "volnay"
  from b in buveurs)

4. CONCLUSION

- **Autres aspects à approfondir :**
  - les méthodes d 'indexation des objets complexes :
    - index de chemins
  - techniques d 'optimisation de requêtes objets :
    - les stratégies d 'évaluation
    - le modèle de coût
  - les vues
  - la gestion des droits
  - les performances

- **L'évolution JDO : Java Data Objects**
  - 2002 : JDO 1.0
  - 2004 : apparition de plusieurs SGBDO libres:
    - db4Objects, Perst, Ozone, Magma, ZOBD (Zope Object Database), EyeDB (with OQL)
  - 2006 : JDO 2.0
  - 2006 Formation du groupe de travail Object Database Technology (ODBT WG) au sein de l'OMG

Exemple d'application : db4o (db4objects by versant)



Exemple d'utilisation : db4o (1)

Classe Personne

```
public class Personne {
  public static class Adresse { ... }

  private String nom;
  private String prenom;
  private List<Adresse> coords;
  private Set<Livre> emprunts;

  public Personne(String nom, String prenom) { ... }

  public String getNom() { return nom; }
  public void setNom(String nom) { this.nom = nom; }
  public String getPrenom() { return prenom; }
  public void setPrenom(String nom) { this.prenom = prenom; }

  public void habite(Adresse a) { coords.add(a); }
  public boolean isHabitantDe(String ville) { ... }

  public void emprunte(Livre l) { emprunts.add(l); }
  public void rend(Livre l) { emprunts.remove(l); }

  public String toString() { ... }
}
```


Exemple d'utilisation : db4o (2)

Classe Livre

```
public class Livre {
  private String titre;
  private String auteur;

  public Livre(String titre, String auteur) { ... }
  public String toString() { return "{" + titre + ", " + auteur + "}"; }
}
```


Exemple d'utilisation : db4o (3)

Création d'instances

```
import com.db4o.*;
import com.db4o.query.*;

public class App {
  static public void main(String[] args) {
    ObjectContainer db = Db4o.openFile("Essai.db4o");
    try {
      Livre l1 = new Livre("Titre 1", "Auteur 1");
      Livre l2 = new Livre("Titre 2", "Auteur 2");
      Livre l3 = new Livre("Titre 3", "Auteur 3");

      Personne p1 = new Personne("Dupont", "Jean");
      p1.habite(new Personne.Adresse("Paris"));
      p1.habite(new Personne.Adresse("Versailles"));
      p1.emprunte(l1);
      p1.emprunte(l2);
      db.set(p1);
```

Exemple d'utilisation : db4o (4)

```
        Personne p2 = new Personne("Martin", "Michel");
        p2.habite(new Personne.Adresse("Clermont"));
        p2.emprunte(l1);
        p2.emprunte(l3);
        db.set(p2);

        Personne p3 = new Personne("Dupont", "Jacques");
        p3.habite(new Personne.Adresse("Versailles"));
        p3.emprunte(l3);
        System.out.println(p3);
        db.set(p3);
      }
      finally {
        db.close();
      }
    }
  }
```


Exemple d'utilisation : db4o (5)

Requêtes de classe : les personnes, les livres

```
static public void classQueryPersonne(ObjectContainer db) {
  List<Personne> personnes = db.query(Personne.class);
  for (Personne p : personnes) {
    System.out.println(p);
  }
}

static public void classQueryLivre(ObjectContainer db) {
  List<Livre> livres = db.query(Livre.class);
  for (Livre l : livres) {
    System.out.println(l);
  }
}
```


Exemple d'utilisation : db4o (6)

Requête QBE : les personnes de nom « Dupont »

```
static public void qbeQuery(ObjectContainer db) {
  System.out.println("QBE Query : les Dupont");
  ObjectSet<Personne> result = db.get(new Personne("Dupont", null));
  for (Personne p : result) {
    System.out.println(p);
  }
}
```


Exemple d'utilisation : db4o (7)

Requête native : les habitants de « Versailles »

```
static public void nativeQuery(ObjectContainer db) {
  List<Personne> versaillais = db.query(new Predicate<Personne>() {
    public boolean match(Personne p) {
      return p.isHabitantDe("Versailles");
    }
  });
  for (Personne p : versaillais) {
    System.out.println(p);
  }
}
```

Mise à jour

```
p1.rend(l1);
db.set(p1);
```