

Persistence objet

JDO

CNAM
2005-2006

Plan de la présentation

- **Introduction**
 - ▼ Persistence
 - ▼ Impedance mismatch
 - ▼ Approche « POJO »
 - ▼ Solution de persistance objet
- **Principes JDO**
 - ▼ Objectfs
 - ▼ Cycle de développement JDO
 - ▼ Classes d'une application JDO
- **Exemple JDO**
- **TP**
 - ▼ Mise en place de du cycle de développement JDO
 - ▼ Intégration de la technologie JDO dans l'application Web *annuaire*

Introduction

La persistance (1)

- **Sauvegarder l'état courant des données manipulées par une application**

- ▼ Tant que les données nécessaires au(x) traitement(s) sont présentes dans les structures du programme (tableaux, listes,...) => données transitoires (*transcientes*)

- ▼ Persistance inutile si la mémoire de travail des ordinateurs :

- n'était pas volatile (nécessitant une alimentation en courant pour être maintenue)
- avait un coût moindre (permettant de l'augmenter à la taille des disques durs typiquement).

La persistance (2)

- **Base de données :**
 - ▼ Ensemble de données formant un ensemble logique cohérent
- **Plusieurs solutions techniques :**
 - ▼ SGBDR (relationnels) : les plus utilisés
 - Oracle
 - PostgreSQL
 - MySQL
 - SQLServer
 - ...
 - ▼ SGBDO (objets) : tentative de réunir le monde des LPOO et les BD.
Années 80-90 mais problèmes d'efficacité et de fiabilité
 - Versant
 - O2
 - Matisse
 - ...
 - ▼ Fichiers plats
 - Texte
 - Tableurs (Excel,...)
 - ▼ XML
 - EDI (Echange de données informatisées)

La persistance (3)

- **SGBD relationnels les plus utilisés**
 - ▼ Le modèle relationnel offre un cadre cohérent pour :
 - Modéliser les données (redondance, abstraction, cohérence, ...)
 - Interroger les données (efficacité : index, logique d'accès : langage SQL normalisé, ...)
 - Mettre à jour les données (gestion des transactions, gestion de la cohérence, résistance aux pannes, ...)
- **SGBD incontournables dans les systèmes d'information**
- **Monde idyllique ?**
 - ▼ Mais nécessité des langages de programmation pour construire les applications informatiques qui accèdent, stockent et mettent à jour les données.

Difficultés

- **Nécessité d'assurer une correspondance (*mapping*) entre les langages de programmation (orientés objets ou non) et les SGBD(R)**
 - ▼ « impedance mismatch » entre le modèle relationnel et le monde des langages de programmation
 - **Types de données différents :**
 - Types relationnels simples (varchar, number, ...)
 - Types LP structurés et imbriqués même si des types simples peuvent correspondre avec les types relationnels (String, int, ...)
 - **Moteurs d'exécution différents**
 - Ensembliste pour le monde relationnel
 - Tuple à tuple pour le monde des LP
- **Des solutions existent qui répondent :**
 - ▼ Aux différences des moteurs d'exécution
 - ▼ Aux différences du typage des données

Etablir un pont entre les deux mondes

- **Des SGBD vers les LP**
 - ▼ SGBD Objets
 - ▼ PL/SQL (Procedural Language SQL) : moteur d'exécution procédural ajouté (tests, boucles, fonctions, procédures, ...)
 - ▼ Moteur d'exécution de langage directement au sein des SGBD (Java pour Oracle, Python dans PostgreSQL, ...)
 - ▼ Types complexes SQL
 - Oracle p.e. : type composés *array*, structurés, sérialisation d'instances d'objet Java dans une table, ...)
 - ▼ ...
- **Des LP vers les SGBD**
 - ▼ Drivers de connexion au SGBD : JDBC, ADO.NET, ODBC, drivers PHP
 - Efficacité
 - API de bas niveau (exigent des compétences dans le domaine des langages ET des SGBD cibles)
 - En dehors des types simples, gestion manuelle de l'impedance mismatch
 - ▼ Systèmes de plus haut niveau offrant une couche d'abstraction dans l'accès à la base de données
 - Solutions de persistance objet (mapping objet/relationnel)

Différentes approches

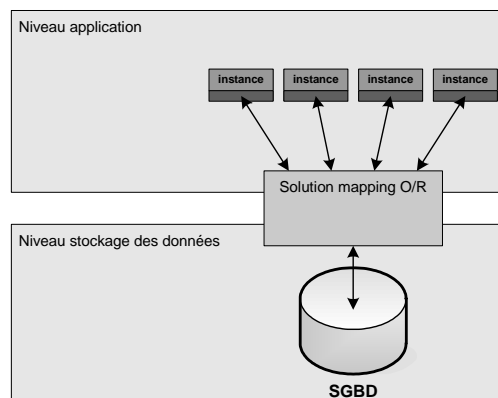
- **Solutions qui impliquent un modèle objet et un modèle de stockage (relationnel ou non)**
- **Trois approches :**
 - ▼ **Top-down :**
 - Le modèle objet existe (ou est imposé), et l'on souhaite générer modèle de stockage.
 - Génération de schéma de base relationnelle à partir d'un ensemble de classes Java.
 - ▼ **Bottom-up :**
 - Le modèle de stockage existe (ou est imposé), et l'on souhaite générer le modèle objet.
 - Génération de classes Java à partir d'un schéma relationnel.
 - Cas du TP « annuaire » : l'application est construite au-dessus du modèle relationnel et les évolutions (création des départements) partent aussi du modèle relationnel
 - ▼ **Meet-in-the-middle :**
 - Le modèle de stockage et le modèle objet existent (ou sont imposés), et l'on souhaite faire correspondre l'un avec l'autre.
 - ▼ **Dans tous les cas, une formalisation de la correspondance (mapping) entre les deux modèles devra être produite.**

Approche « POJO »

- **POJO : *Plain Old Java Objects***

- ▼ **Bons vieux objets Java tous simples**

- **Utiliser un modèle objet simple (facile à développer, centré sur l'applicatif plus que sur le technique) et sans dépendances avec une API ou un framework**
- **Approche top-down, l'application est conçue sans prendre en compte le monde relationnel :**
 - ▼ On réalise une modélisation objet
 - ▼ La persistance est assurée par une solution tierce.



Solutions Java pour la persistance objet

- **Enterprise Java Beans**
 - ▼ EJB CMP (Container Manage Persistence)
 - ▼ EJB est d'abord un modèle de composants métiers et est plus limité dans le domaine de la persistance d'objets
 - ▼ Connue pour sa lourdeur
 - ▼ Difficile à maintenir
 - ▼ Amélioration avec les futurs EJB 3
- **Hibernate**
 - ▼ JBoss (serveur d'applications)
 - ▼ Très utilisé
- **JDO (Java Data Object)**
 - ▼ Similaire à Hibernate dans ses objectifs
 - ▼ Spécification de Sun (Java Specification Request 12)
 - ▼ Version 1.0 en 2002
 - ▼ Maturité avec la version 2.0 en mars 2005
 - ▼ Depuis, nombreuses implémentations de la spécification et JDO devient un standard dans les solutions de gestion de la persistance en Java.

JDO

Principes

Objectifs de JDO

- **Les principaux buts de JDO sont :**
 - ▼ **La facilité d'utilisation**
 - Modélisation objet sans contrainte : supporte l'héritage, les collections,...
 - Gestion automatique du mapping des données
 - ▼ **La persistance universelle :**
 - Persistance vers tout type de systèmes de gestion de ressources (bases de données relationnelles, fichiers, ...). C'est au déploiement de l'application que la liaison avec la cible sera faite.
 - JDBC est limité au SGBDR, JDO non (en théorie, dépend de l'implémentation)
 - ▼ **La transparence vis à vis du système de gestion de ressources utilisé :**
 - Ce n'est plus le développeur mais JDO qui dialogue avec le système de gestion de ressources (ne nécessite pas la connaissance du schéma de la base, pas de requêtes SQL,...)
 - ▼ **La standardisation des accès aux données**
 - ▼ **La prise en compte des transactions**

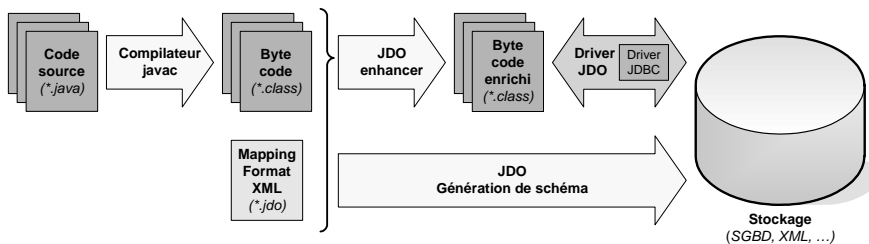
Spécification JDO

- **JDO est une spécification qui définit un standard :**
 - ▼ **Pour pouvoir l'utiliser il faut utiliser une implémentation fournie par un fournisseur (comme les spécifications J2EE : JDBC, EJB, JavaMail, ...).**
 - ▼ **L'intérêt des spécifications est qu'il est possible d'utiliser le même code avec des implémentations différentes tant que l'on utilise uniquement les fonctionnalités précisées dans les spécifications**
 - ▼ **Plusieurs implémentations existent :**
 - XCalia LIDO (commercial)
 - JPox (solution libre) :
 - Sélectionné par Sun pour devenir l'implémentation de référence de JDO (comme Tomcat l'est pour la techno JSP/Servlets)
 - Limité pour le moment à la persistance vers les SGBDR
 - Castor JDO (solution libre)
 - ...
 - ▼ **Chaque implémentation est capable d'utiliser un ou plusieurs systèmes de stockage de données particulier (base de données relationnel, base de données objets, fichiers, ...).**

Cycle de développement avec JDO (1)

- **Le développement avec JDO se déroule en plusieurs étapes :**
 - ▼ **Après conception objet, écriture des classes Java destinées à stocker les données (codage Java tout ce qui a de plus *classique*) :**
 - Objets qui deviendront persistants par la suite
 - ▼ **Écriture du fichier metadata (au format XML) qui précise le mapping entre les objets et le système de gestion des ressources (SGBD ou autre)**
 - ▼ **Enhancement (enrichissement) des objets destinés à devenir persistant**
 - Enrichissement qui doit être portable d'une implémentation JDO à l'autre
 - ▼ **Génération du schéma dans le système de stockage**
 - ▼ **Écriture des objets qui utilisent les objets métiers pour répondre aux besoins fonctionnels. Ces objets utilisent l'API JDO.**

Cycle de développement avec JDO (2)



Classes d'une application JDO

■ Trois type de classes :

▼ Non-*enhancées*

- N'ayant aucun lien avec JDO
- RAS

▼ *Persistence capable* :

- Les classes d'objets capables de persister leurs données et/ou de les modifier transactionnellement.
- Ces classes n'ont rien de particulier, si ce n'est le fait que leur code a été enrichi après leur compilation.

▼ *Persistence aware* :

- Les classes conscientes de la persistance des *persistence capable* : ce sont les objets métiers qui assurent le rôle fonctionnel de l'application
- Ils utilisent l'API JDO pour les manipuler :
 - requêtes de recherches multi-critères, délimitation des transactions, changement d'état JDO, etc.).

Classes *persistence capable* (1)

■ Objet Java capable d'être :

▼ Transactionnel :

- Les modifications depuis le dernier début de transaction peuvent être annulées (l'état précédent est rétabli)

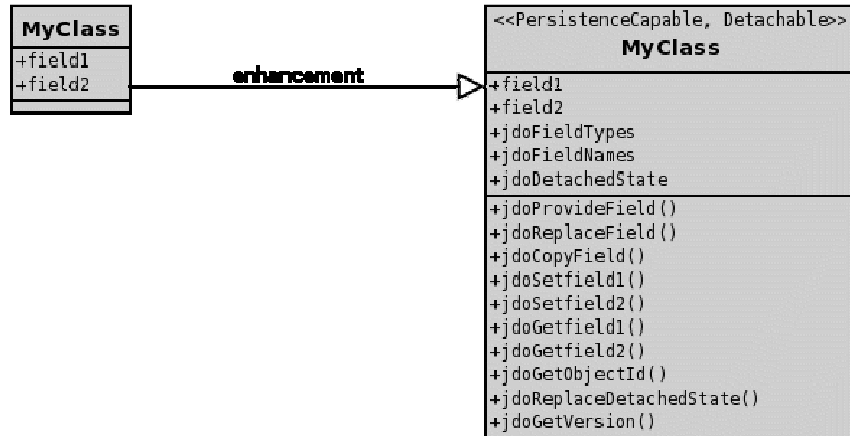
▼ Persistant de manière transparente (sans que le développeur ait à modifier son code source).

■ Cette capacité acquise après modification de son code :

▼ Modification du code par un outil (*enhancer*)

- Modification du code après compilation appelée *enhancement* (enrichissement de code).
- Généralement c'est le code compilé qui est enrichi (c'est le cas de JPox)
- Cet *enhancement* se base sur un descriptif des classes à rendre *persistence capable* (définissant les classes et champs à rendre persistants, les champs caractérisant l'identité d'un objet, etc.), nommé métadonnées (metadata).

Classes *persistence capable* (2)



Classes *persistence aware*

■ Les classes manipulant les objets *persistence capable* effectuent des :

▼ lectures :

- Recherche par identifiant unique
- Recherche multi-critères (requête JDO QL), itération sur les résultats, tris, etc.
- lectures partielles (groupe de champs chargés par défaut, et non pas tous les champs d'un objet)
- Lectures non transactionnelles : plus performantes mais pouvant être incohérentes (un attribut lu n'est pas garanti cohérent avec un autre lu précédemment par exemple)

▼ Ecritures : modification de l'état

- Automatiquement répercuté en base lors de la validation de la transaction
- Répercuté uniquement dans le cache, en cas d'écriture non transactionnelle.

API JDO

- L'API de JDO est définie dans le package `javax.jdo`.
- Les classes d'objets potentiellement persistant et/ou transactionnels.
 - ▼ Elles implémenteront l'interface *PersistenceCapable* suite à leur post-compilation (*enhancement*).
 - ▼ Tout ou partie de leur champs (selon le mapping spécifié dans le fichier metadata) sera alors pris en charge par JDO à l'exécution.
- Les classes conscientes de la persistance (persistence aware), utilisant l'API JDO pour manipuler des objets *PersistenceCapable*
 - ▼ Ces classes utilisent typiquement des *PersistenceManager*, aptes à :
 - Exécuter des requêtes
 - Démarrer/terminer des transactions
 - Changer l'état des objets JDO (persistant, transient, transactionnel, etc.)
 - ▼ Les *PersistenceManager* sont obtenus via une *PersistenceManagerFactory*.

JDO

Exemple

JPOX

■ Utilisation de JPOX :

- ▼ <http://www.jpox.org/index.jsp>
- ▼ *JPOX is a free and fully compliant implementation of the JDO 1.0 and 2.0 specifications*
- ▼ Support de la persistance pour la plupart des SGBD
- ▼ Requêtes via JDOQL ou SQL
- ▼ JPOX 1.0 implémente la spécification JDO 1.0
- ▼ JPOX 1.1 implémente la spécification JDO 2.0
 - Encore en cours de développement

Exemple JDO

■ Assurer la persistance d'instances de livres :

- ▼ Classe unique *Livre* :
 - Titre
 - Auteur
 - Résumé
 - Prix

Classe Livre

```
package tp2cnam;

public class Livre
{
    public String titre;
    public String auteur;
    public String resume;
    public int    prix;

    /**
     * Constructeur par défaut, obligatoire avec JDO pour une classe que l'on veut
     * rendre persistante.
     */
    public Livre() { }

    public Livre(String titre,String auteur,String resume,int prix)
    {
        this.titre = titre;
        this.auteur = auteur;
        this.resume = resume;
        this.prix = prix;
    }

    public String toString()
    {
        return "Livre ["+titre+", "+auteur+", "+resume+", "+prix+"];";
    }
}

```

Livre.java

Javac Livre.java produit Livre.class

Gestion du mapping

```
<?xml version="1.0"?>
<!DOCTYPE jdo PUBLIC
"-//Sun Microsystems, Inc.//DTD Java Data Objects Metadata 2.0//EN"
"http://java.sun.com/dtd/jdo_2_0.dtd">

<jdo>
  <package name="tp2cnam">
    <class name="Livre" identity-type="datastore">
      <inheritance strategy="new-table"/>

      <field name="titre" persistence-modifier="persistent">
        <column length="30" jdbc-type="VARCHAR"/>
      </field>

      <field name="auteur" persistence-modifier="persistent">
        <column length="30" jdbc-type="VARCHAR"/>
      </field>

      <field name="resume" persistence-modifier="persistent">
        <column length="1000" jdbc-type="VARCHAR"/>
      </field>

      <field name="prix" persistence-modifier="persistent">
        <column length="10" jdbc-type="NUMBER"/>
      </field>

    </class>
  </package>
</jdo>

```

Package.jdo

Gestion du mapping (plus simple)

```
<?xml version="1.0"?>
<!DOCTYPE jdo PUBLIC
"-//Sun Microsystems, Inc.//DTD Java Data Objects Metadata 2.0//EN"
"http://java.sun.com/dtd/jdo_2_0.dtd">
```

Package.jdo

```
<jdo>
<package name="tp2cnam">
  <class name="Livre">

    <field name="titre" />

    <field name="auteur" />

    <field name="resume" />

    <field name="prix" />

  </class>
</package>
</jdo>
```

- **Correct mais le typage en base de données (Oracle) se fait avec des valeur par défaut :**

- ▼ Type *String* Java :
 VARCHAR(255)

- ▼ Type *int* Java :
 NUMBER(10)

- ▼ Etc.

Enhancement du code

- **Appel de l'outil *JPOXEnhancer* :**

```
java -classpath .;..\lib\bcel-5.1.jar;
      ..\lib\jdo2-api-2.0-rc1.jar;
      ..\lib\jpox-1.1.0-rc-1.jar;
      ..\lib\jpox-enhancer-1.1.0-rc-1.jar;
      ..\lib\log4j.jar
      -Dlog4j.configuration=file:log4j.properties
      org.jpox.enhancer.JPOXEnhancer
      tp2cnam\package.jdo
```

- **La classe compilée *Livre.class* est modifiée**

- ▼ **La seule modification visible est la taille du fichier**

Génération du schéma (1)

■ Appel de l'outil *SchemaTool* :

```
java -classpath .;..\lib\jdo2-api-2.0-rc1.jar;  
        ..\lib\jpx-1.1.0-rc-1.jar;  
        ..\lib\log4j.jar;  
        ..\lib\classes12.jar  
-Djavax.jdo.option.ConnectionDriverName=  
  oracle.jdbc.driver.OracleDriver  
-Djavax.jdo.option.ConnectionURL=  
  jdbc:oracle:thin:USER/PASSWORD@localhost:1521:BASE  
-Dlog4j.configuration=file:log4j.properties  
org.jpox.SchemaTool  
tp2cnam\package.jdo -create
```

■ Création des tables :

▼ JPOX_TABLES

- Référencement des tables générées par JPox

▼ LIVRE

- La table qui assurera la persistance des instances de *Livre* créées par l'application.

Génération du schéma (2)

```
desc JPOX_TABLES;
```

Name	Null?	Type
CLASS_NAME		NOT NULL VARCHAR2(128)
TABLE_NAME	NOT NULL	VARCHAR2(128)
TYPE	NOT NULL	VARCHAR2(4)
OWNER	NOT NULL	VARCHAR2(2)
VERSION	NOT NULL	VARCHAR2(20)

```
desc LIVRE;
```

Name	Null?	Type
LIVRE_ID	NOT NULL	NUMBER
AUTEUR		VARCHAR2(255)
PRIX	NOT NULL	NUMBER(10)
RESUME		VARCHAR2(255)
TITRE		VARCHAR2(255)

Insertion de données

```
public void creationLivres()
{
    Transaction tx = pm.currentTransaction();
    try
    {
        tx.begin();
        Livre l1 = new Livre("JSP","Philippe TANGUY","Pas terrible",7);
        Livre l2 = new Livre("JDO","Philippe TANGUY","Bof...",10);

        pm.makePersistent(l1);
        pm.makePersistent(l2);
        tx.commit();
    }
    finally
    {
        if (tx.isActive())
        {
            tx.rollback();
        }
        pm.close();
    }
    System.out.println("Données créées.");
}
```

Stockage dans la base

```
SQL> select * from LIVRE;
```

LIVRE_ID	AUTEUR	RIX	RESUME	TITRE
1	Philippe TANGUY	7	Pas terrible	JSP
2	Philippe TANGUY	10	Bof...	JDO