Realizzare la persistenza di un modello ad oggetti tramite JDO

Sandro Pinna
Agile Group - DIEE - Università degli studi di Cagliari

Definizione di JDO

- JDO (Java Data Objects) definisce una interfaccia (API) per la persistenza di oggetti java in un datastore.
- JDO è indipendente dal datastore. Stessa interfaccia per realizzare la persistenza su RDBMS, OODBMS, XML, File System..
- JDO è uno standard (versione attuale 2.0)
- Esistono diverse implementazioni di JDO (commerciali, FLOSS). JDO definisce le interfacce che devono essere implementate dai produttori.

Persistenza di oggetti

- Oggetto = istanza di una classe
- Application Object Model (collezione di tutte le classi dell'applicazione)
- MVC (Model, View, Controller)
- Domain Object Model (insieme delle classi che definiscono il business domain dell'applicazione)
- Object persistence. Gli oggetti risiedono in memoria. La persistenza permette di allungare il tempo di vita di un oggetto.

Meccanismi di persistenza

- RDMS (Relational Database Management System) con JDBC ed SQL
- File System
- OODMS
- JDO

JDO: tipi di classi

- JDO suddivide le classi in tre tipi. Il tipo di una classe definisce il modo in cui essa interagisce con JDO:
  - **Persistence Capable** classes (sono classi le cui istanze possono essere rese persistenti sul datastore). Devono essere “arricchite” sulla base delle informazioni presenti in un apposito file xml (JDO Metadata File).
  - **Persistence aware** classes (hanno attibuti di tipo Persistence Capable).
  - **Normal** classes.

Esempio di file .jdo

Vedere eclipse...

PersistenceManagerFactory

```
Properties props= new Properties();
props.put("javax.jdo.PersistenceManagerFactoryClass",
"org.jpox.PersistenceManagerFactoryImpl");
props.put("javax.jdo.option.ConnectionDriverName",
"com.mysql.jdbc.Driver");
props.put("javax.jdo.option.ConnectionURL",
"jdbc:mysql://localhost/jpox");
props.put("javax.jdo.option.ConnectionUserName",
"mysql");
props.put("javax.jdo.option.ConnectionPassword","");
PersistenceManagerFactory pmf=
JDOHelper.getPersistenceManagerFactory(props);
```

PersistenceManager

- La classe PersistenceManager permette la gestione delle operazioni di memorizzazione, modifica, cancellazione degli oggetti.
- PersistenceManager pm = pmf.getPersistenceManager();

Rendere un oggetto persistente

```
MyClass my_obj= ....;
Transaction tx;
try{
    tx = pm.currentTransaction();
    tx.begin();
    pm.makePersistent(my_obj);
    tx.commit();
}
catch (Exception e){
    if (tx.isActive()){
        tx.rollback();
    }
}
```

Caricare in memoria un oggetto persistente

```
try{
    tx = pm.currentTransaction();
    tx.begin();
    Extent e=pm.getExtent(mydomain.MyClass.class,true);
    Iterator iter=e.iterator();
    while (iter.hasNext()){
        MyClass my_obj=(MyClass)iter.next();
        ...
    }
    tx.commit();
}catch (Exception e){
    if (tx.isActive()){
        tx.rollback();
    }
}
```

Cancellare un oggetto dal datastore

```
try{
    tx = pm.currentTransaction();
    tx.begin();
... (code to retrieve object in question) ...

    pm.deletePersistent(my_obj);

    tx.commit();
}
catch (Exception e){
    if (tx.isActive())
    {
        tx.rollback();
    }
}
```

Rendere un oggetto transiente

pm.makeTransient(myObj);
Deprecated in jdo 2.0

Attach- Detach (1/ 2)

```
// Detach the object from the datastore
try{
   tx = pm.currentTransaction();
   tx.begin();
   ... (code to retrieve object in question) ...

   detachedObj = pm.detachCopy(my_obj);
   tx.commit();
}
catch (Exception e){
   if (tx.isActive())
   {
      tx.rollback();
   }
}
```

Attach- Detach (2/ 2)

```
... (code to work on "detachedObj")
// Attach the updated object to the datastore
try{
    tx = pm.currentTransaction();
    tx.begin();
    pm.attachCopy(my_obj, true);
    tx.commit();
}
catch (Exception e){
    if (tx.isActive())
    {
        tx.rollback();
    }
}
```

Lo stato degli oggetti in JDO

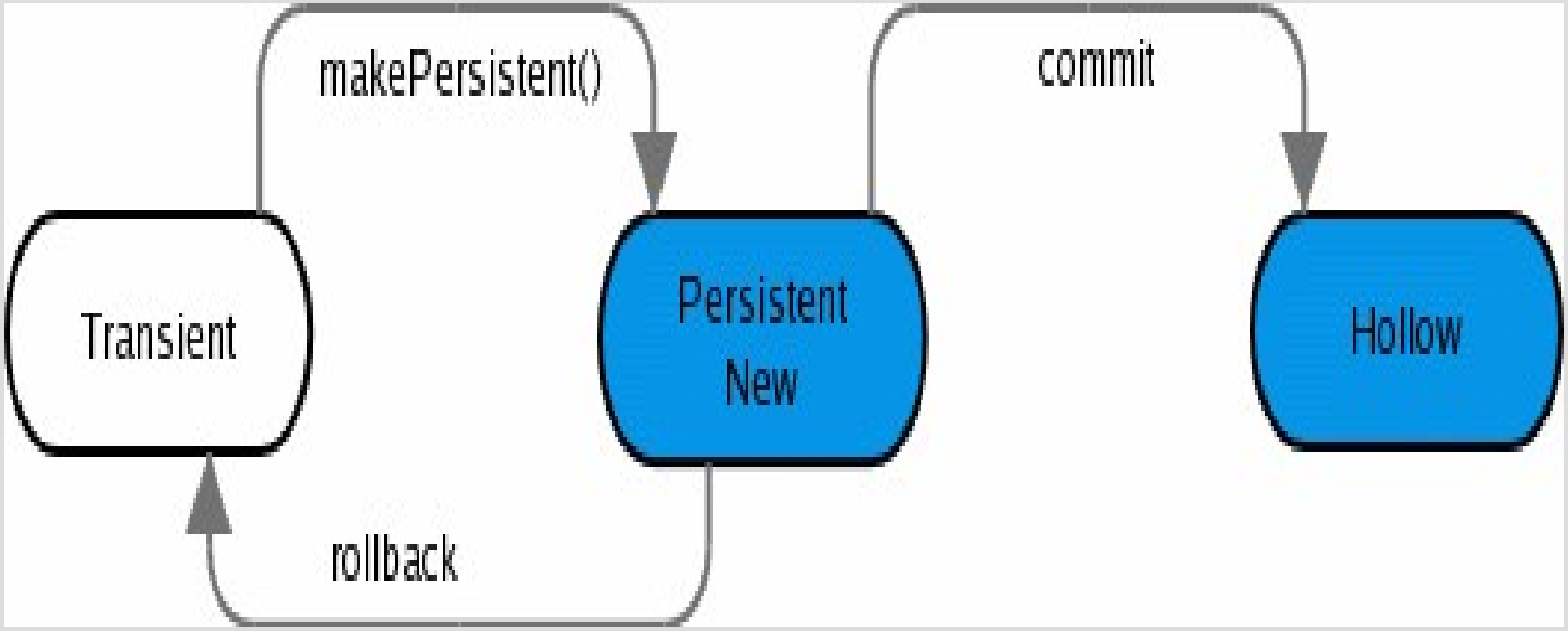
JDO gestisce il ciclo di vita di un oggetto, dalla creazione (Transient) alla persistenza nel datastore (Hollow, Persistent Clean) passando per tutti gli altri stati intermedi:

- Transient
- Persistent New
- Persistent Dirty
- Hollow
- Persistent Clean
- Persistent Deleted
- Persistent New Deleted...

Rendere persistente un oggetto: il codice

```
Transaction tx=pm.currentTransaction();
try{
    tx.begin();
    Product product=new Product("...",".....",49.99);
    pm.makePersistent(product);
    tx.commit();
}finally{
    if (tx.isActive())
    {
        tx.rollback();
    }
}
```

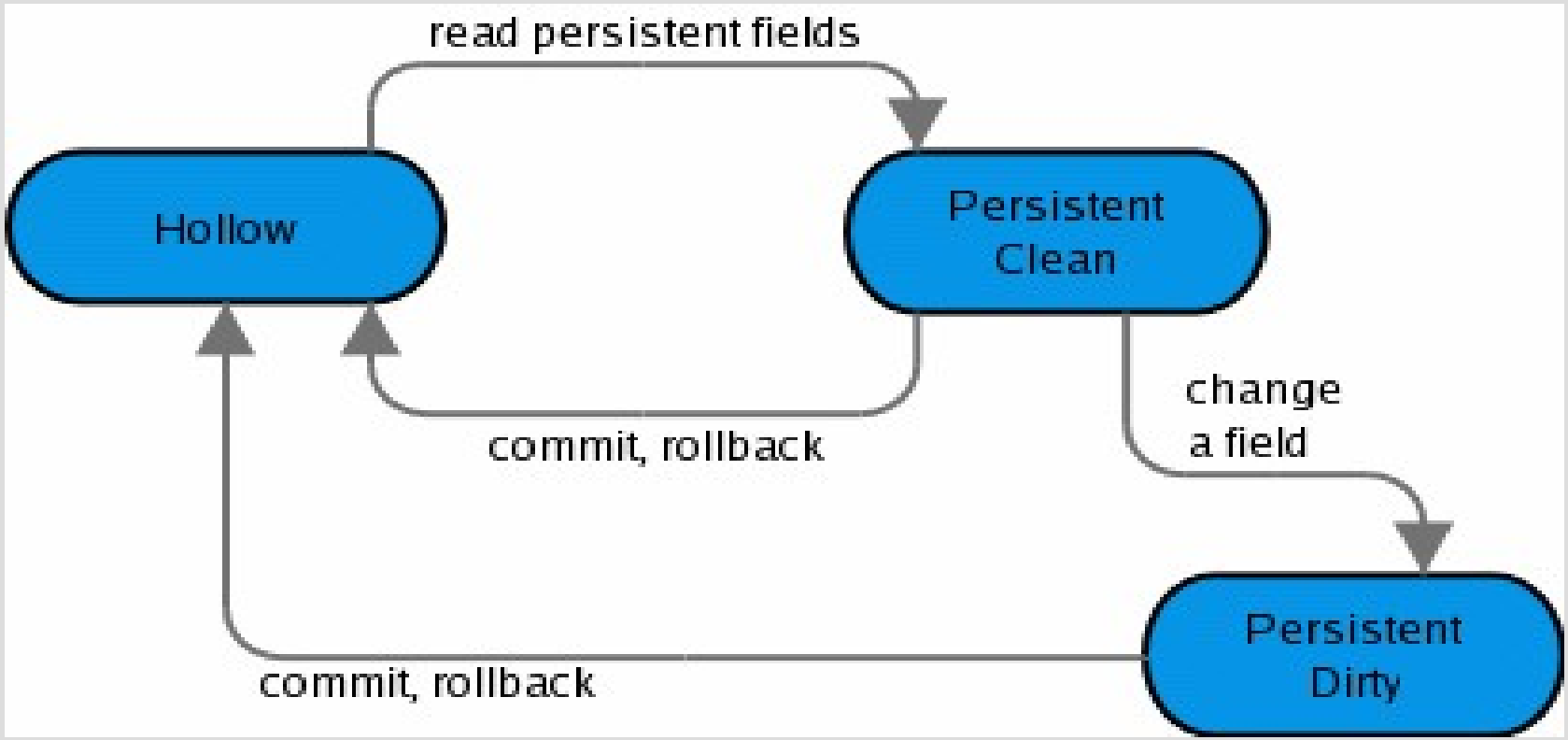
Rendere persistente un oggetto: gli stati



Aggiornare un oggetto: il codice

```
Transaction tx=pm.currentTransaction();
try{
    tx.begin();
    String product_name=product.getName();
    ...
    product.setPrice(75.0);
    tx.commit();
}finally{
    if (tx.isActive())
    {
        tx.rollback();
    }
}
```

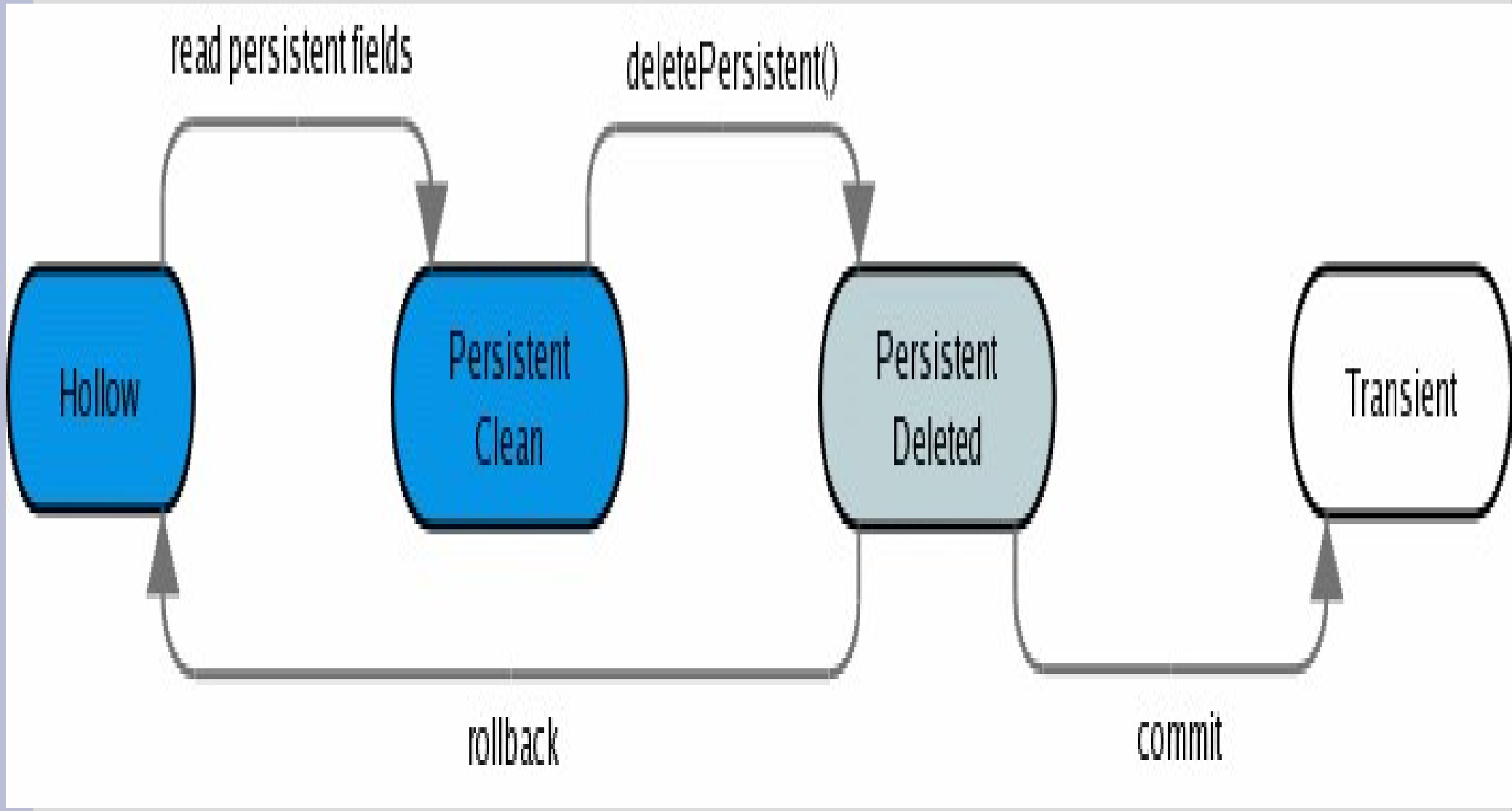
Aggiornare un oggetto: gli stati



Delete

```
Transaction tx=pm.currentTransaction();
try{
    tx.begin();
    String product_name=product.getName();
    ...
    pm.deletePersistent(product);
    tx.commit();
}finally{
    if (tx.isActive())
    {
        tx.rollback();
    }
}
```

Delete


read persistent fields
deletePersistent()
Hollow
Persistent Clean
Persistent Deleted
Transient
rollback
commit

Query con JDOQL

```
Query query = pm.newQuery("javax.jdo.query.JDOQL",
  "SELECT FROM org.jpox.MyClass WHERE param2 <
  threshold");
query.declareImports("import java.util.Date");
query.declareParameters("Date threshold");
query.setOrdering("param1 ascending");
Collection results =
  (Collection)query.execute(my_threshold);
```

Named Query (solo jdo2.0)

```
<![CDATA[
<jdo>
   <package name="org.jpox.example">
      <class name="Product">
         ...
         <query name="SoldOut"
  language="javax.jdo.query.JDOQL"><![CDATA[
         SELECT FROM org.jpox.example.Product WHERE
  status == "Sold Out"
         ]]></query>
      </class>
   </package>
</jdo>]]>
```

Named Query (solo jdo 2.0)

```
Query query = pm.newNamedQuery(
org.jpox.example.Product.class,"SoldOut");
Collection results=(Collection)query.execute();
```

Object Identity

- L'identità di un oggettoè lo strumento attraverso il quale JDO identifica univocamente un oggetto.
- L'identità può essere gestita dal sistema (Identity Datastore). Il sistema decide quando due oggetti sono uguali.
- L'alternativa è: Application Identity. Inquesto caso lo sviluppatore deve definire esplicitamente quando due oggetti sono uguali.

Application Identity

Esempiosu Eclipse

Eccezioni

- JDO definisce le eccezioni per il controllo delle situazioni di errore che possono verificarsi a vari livelli.
- Si tratta di “Runtime Exceptions”.
- Il programamtore è libero di catturare solo le eccezioni che gli interessano.

Implementazioni JDO

JPOX
(www.jpox.org)

Libro su JDO (free):
Java Data Objects, Robin M.Roos