

Technologie odwzorowania obiektowo-relacyjnego: JDO

Plan prezentacji

- Wprowadzenie do JDO
- Trwałość obiektów
- Odwzorowanie obiektowo-relacyjne
- Transakcje i zapytania

Wprowadzenie do JDO

Wprowadzenie

- **JDO** (Java Data Objects) to specyfikacja interfejsu do utrwalania obiektów Java (**POJO** - Plain Old Java Objects) w składnicy danych, którą może być:
 - relacyjny system zarządzania bazą danych,
 - obiektowy system zarządzania bazą danych,
 - pliki tekstowe, pliki XML.
- JDO zostało stworzone przez firmę Sun, wersja 1.0 pojawiła się w 2002 roku, wersja 2.0 w 2005, standard rozwijany pod auspicjami Java Community Process jako JSR-12.
 - <http://java.sun.com/products/jdo/>
 - <http://www.jcp.org/en/jsr/detail?id=12>
 - <http://www.jdocentral.com/>

Kluczowe aspekty JDO

- Podział klas:
 - **zdolne do trwałości** (persistence capable): każda klasa której obiekty mają być utrwalane w składnicy danych musi być uprzednio ulepszona (enhanced) na podstawie metadanych,
 - **świadome trwałości** (persistence aware): klasy jawnie manipulujące składowymi obiektów zdolnych do trwałości, automatycznie ulepszone,
 - **zwyczajne**: nie wymagają metadanych i nie są ulepszone.
- Kontrola trwałości obiektów i zmiany w cyklu życia obiektów
- Wydawanie zapytań do składnicy danych
- Zarządzanie transakcjami

JDO i inne techniki odwzorowania O/R

cecha	JDBC	O/R	EJB ₂	JDO
standardowy i przenaszalny	tak	nie	tak	tak
wybór składnic danych	nie	tak	tak	tak
obsługa POJO	tak	tak	nie	tak
automatyczna implementacja trwałości	nie	tak	nie	tak
do wykorzystania w aplikacji typu stand-alone	tak	tak	nie	tak
do wykorzystania w aplikacji J2EE	tak	tak	tak	tak
możliwe testowanie jednostkowe	tak	tak	nie	tak
dynamiczne zapytania	tak*	tak	nie	tak
generacja kluczy podstawowych	tak*	tak	nie*	tak
wsparcie obiektów dziedziczonych	tak*	tak	nie*	tak
automatyczna generacja schematu	nie	tak	nie*	tak
możliwość wykorzystania istniejącego schematu	tak	tak	tak	tak

Implementacje

- Aktualnie jest dostępnych co najmniej kilkanaście niezależnych implementacji JDO
 - komercyjne: JDO Genie, JPower Map, JCredo, ObjectDB, XCalia, JORM
 - open-source: JDOInstruments, JDOMax, Speedo, Jakarta OJB, JPOX
- JPOX (www.jpox.org) w wersjach 1.0 i 1.1 to rozprowadzana na licencji Apache 2 implementacja standardów JDO1 i JDO2
 - darmowa implementacja open-source,
 - wsparcie większości relacyjnych systemów zarządzania bazami danych (MySQL, Oracle, Sybase, HSQL, PostgreSQL, DB2, Firebird, SAPDB/MaxDB, Informix),
 - umożliwia szerokie wykorzystanie mechanizmu wtyczek,
 - dobrze udokumentowane i wsparte dużą liczbą programistów.

Trwałość obiektów

Interfejs PersistenceManagerFactory

- **PersistenceManagerFactory**: podstawowy obiekt służący do uzyskiwania dostępu do zarządcy trwałości
 - inicjalizacja na poziomie aplikacji
 - inicjalizacja za pomocą parametrów w pliku własności

jdo.properties

...

```
javax.jdo.PersistenceManagerFactoryClass =  
    org.jpox.PersistenceManagerFactoryImpl  
javax.jdo.option.ConnectionDriverName =  
    oracle.jdbc.OracleDriver  
javax.jdo.option.ConnectionURL =  
    jdbc:oracle:thin:@miner.cs.put.poznan.pl:1521:miner10g  
javax.jdo.option.ConnectionUserName = username  
javax.jdo.option.ConnectionPassword = password
```

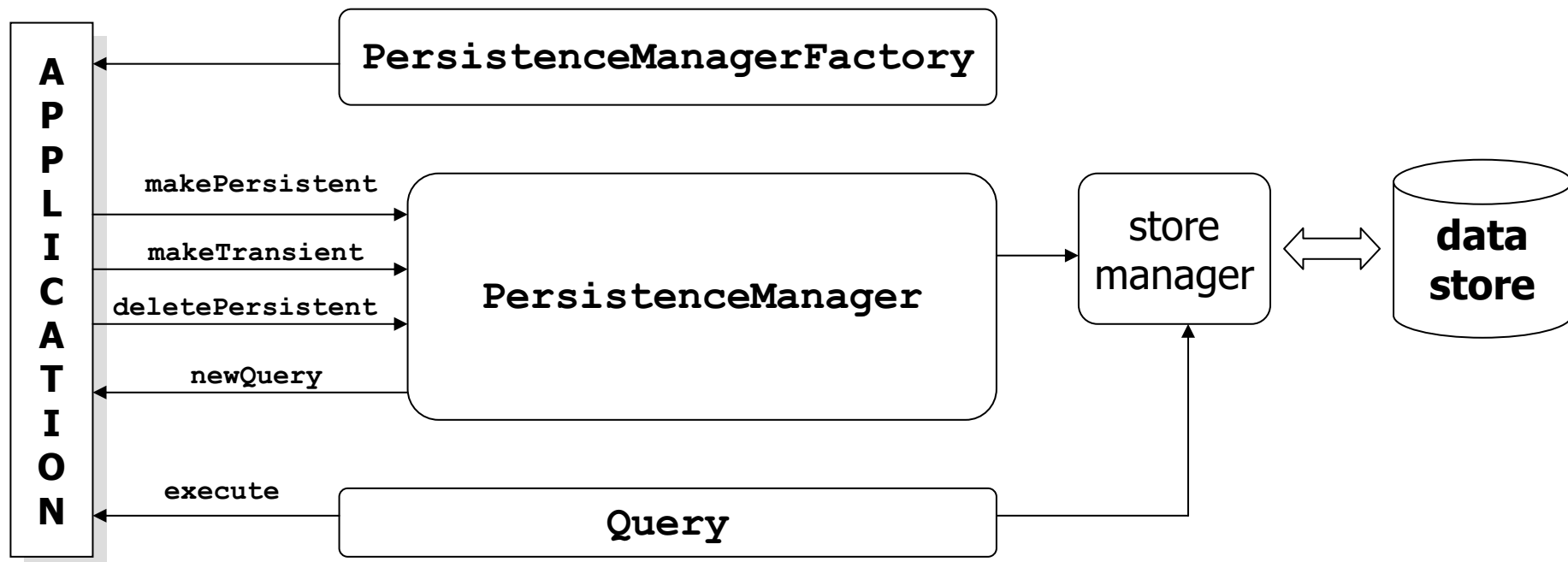
```
File f = new File("jdo.properties");  
PersistenceManagerFactory pmf = new JDOHelper.getPersistenceManagerFactory(f);
```

Interfejs PersistenceManager

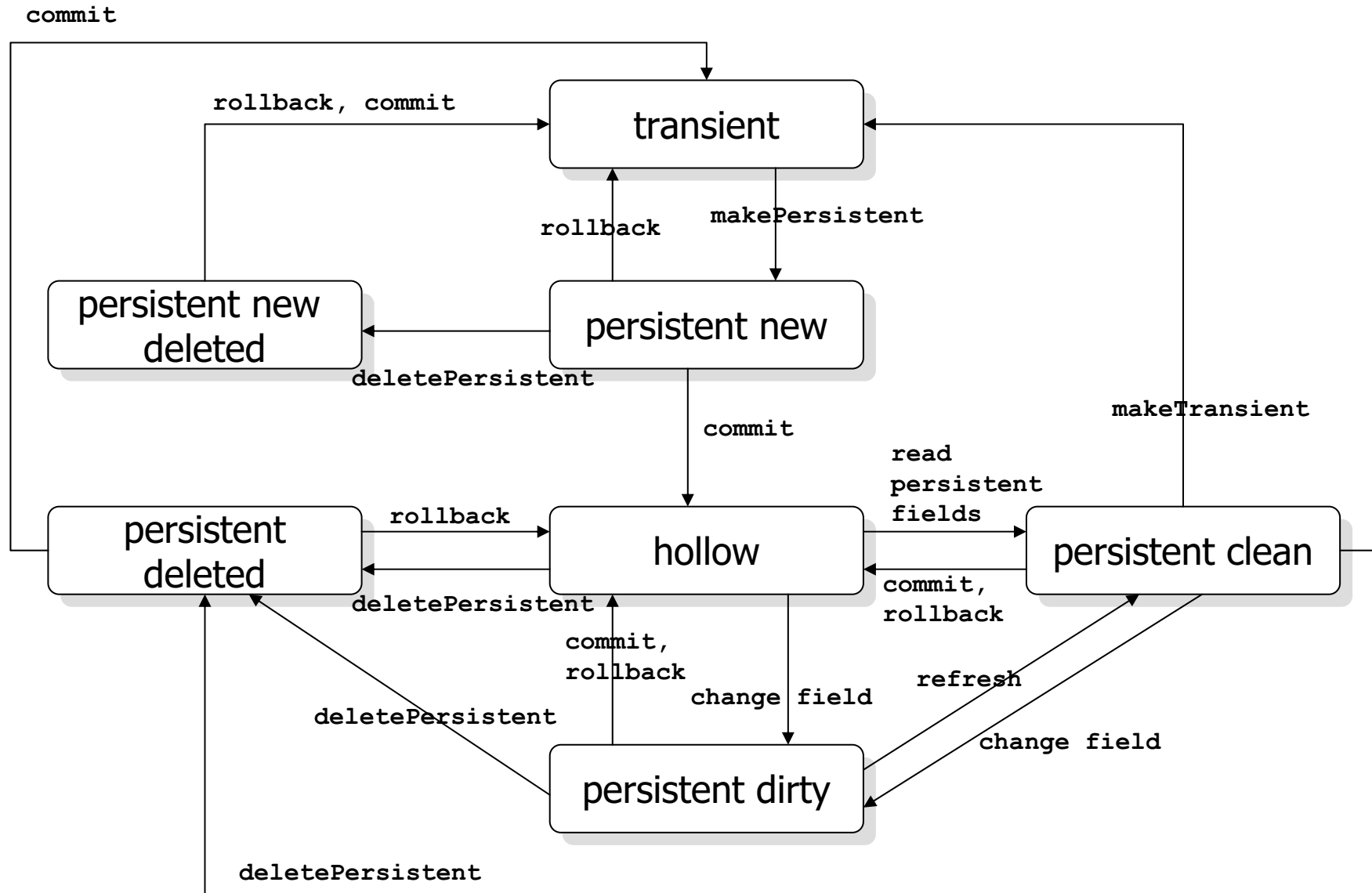
- **PersistenceManager**: podstawowy obiekt służący do utrwalania obiektów, odczytywania trwałych obiektów ze składnicy danych, usuwania obiektów ze składnicy danych

...

```
PersistenceManagerFactory pmf = new JDOHelper.getPersistenceManagerFactory(f);  
PersistenceManager pm = pmf.getPersistenceManager();
```



Stany obiektu



Trwałość obiektów

- Utrwalenie obiektu

```
Employee e = new Employee(...);  
pm.makePersistent(e);
```

- Usunięcie obiektu

```
Employee e = pm.getObjectById(id);  
pm.deletePersistent(e);
```

- Odłączenie obiektu

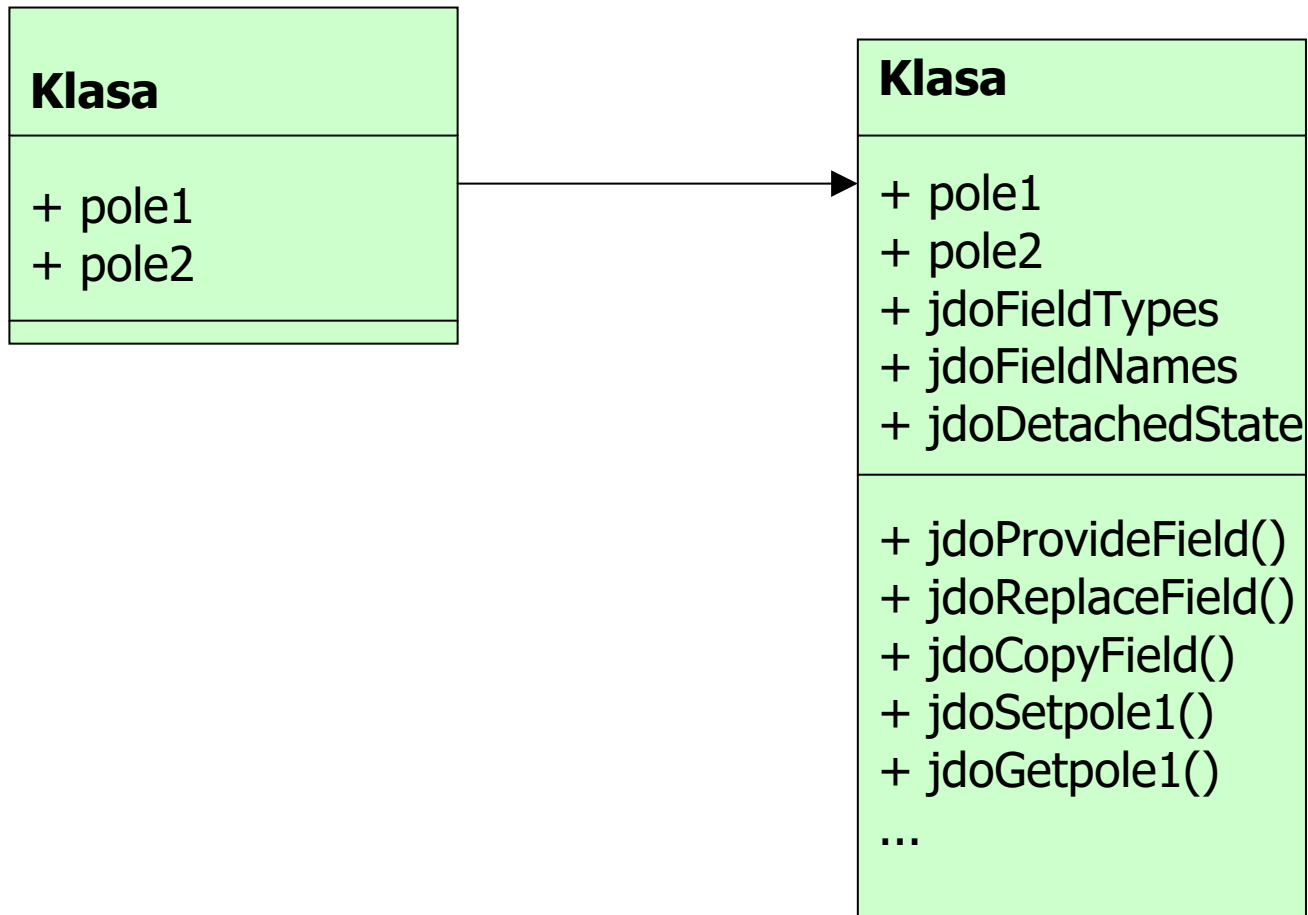
```
Employee ee = pm.detachCopy(e);
```

- Dołączenie obiektu

```
Employee eee = pm.makePersistent(ee);
```

Ulepszanie klas

- Ulepszanie klasy polega na dodaniu do skompilowanego bajtkodu dodatkowych pól i metod wg poniższego schematu:



Ulepszanie klas

- Ręczne ulepszenie klas

```
java -cp classpath org.jpox.enhancer.JPOXEnhancer [options] [jdo-files]
  where options can be
    -d target-dir-name : Write the enhanced classes to the specified dir
    -checkonly : Just check the classes for enhancement status
    -verify : Verify the classes
    -v : verbose output

c:\> java -cp target\classes;lib\jpox-enhancer.jar;lib\jpox.jar;lib\jdo.jar;
lib\log4j.jar;lib\bcel.jar
-Dlog4j.configuration=file:log4j.properties org.jpox.enhancer.JPOXEnhancer
-v target/classes/pl/org/ploug/szkola2006/jdo/package.jdo
```

- Automatyczne ulepszanie klas (!)
 - Maven
 - Ant

Możliwości projektowania

- Istnieją trzy możliwości projektowania aplikacji wykorzystującej JDO:
 - **forward mapping**: dla istniejącego modelu klas projektowany jest schemat składnicy danych,
 - **reverse mapping**: dla istniejącego schematu składnicy danych projektowany jest model klas,
 - **meet-in-the-middle mapping**: istniejący model klas jest dopasowywany do istniejącego schematu składnicy danych.
- Lokalizacja metadanych
 - `pl.org.ploug.szkoła2006`

```
META-INF/package.jdo
WEB-INF/package.jdo
package.jdo
pl/package.jdo
pl/org/package.jdo
pl/org/ploug/package.jdo
pl/org/ploug/szkoła2006.jdo
pl.jdo
pl/org.jdo
pl/org/ploug.jdo
```

Tryby pracy

	nowy schemat	istniejący schemat
tożsamość zarządzana przez składnicę	JDO generuje potrzebne tabele i dodaje atrybut klucza podstawowego oraz generuje identyfikatory obiektów	metadane definiują odwzorowanie tabel i atrybutów, projektant generuje identyfikatory obiektów
tożsamość zarządzana przez aplikację	JDO generuje potrzebne tabele, projektant generuje identyfikatory obiektów	JDO odwzorowuje klasy na istniejące tabele, projektant generuje identyfikatory obiektów

Przykład metadanych

```
package pl.org.ploug;  
  
public class Employee {  
    String ename = null;  
    String job = null;  
    ...  
}
```

package.jdo

```
<?xml version="1.0" encoding="UTF-8"?>  
<!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="pl.org.ploug">  
    <class name="Employee" identity-type="datastore">  
      <field name="ename"/>  
      <field name="job"/>  
    </class>  
  </package>  
</jdo>
```

Tożsamość zarządzana przez składnicę

- W sytuacji gdy tożsamością obiektów zarządza składnica danych, obiekty trwałych klas nie posiadają jawnej składowej zawierającej identyfikator. Identyfikator jest przechowywany tylko po stronie składnicy danych.
- Możliwości generowania identyfikatorów:
 - natywne,
 - inkrementalny,
 - automatycznie przydzielany,
 - pobierany z sekwencera,
 - UUID / hex UUID,
 - maksymalny.

```
<class name="Employee" identity-type="datastore">  
  <datastore-identity strategy="sequence" sequence="nazwa sekwencji"/>  
  ...  
</class>
```

package.jdo

Tożsamość zarządzana przez aplikację

- W przypadku zarządzania tożsamością przez aplikację konieczne jest zdefiniowanie klasy klucza podstawowego. Wartość klucza podstawowego jest przechowywana w składowej danej klasy. Klasa prostego klucza podstawowego jest wbudowana.
- Możliwości generowania identyfikatorów jak poprzednio.

```
<class name="Employee" identity-type="application">  
  <field name="empNo" primary-key="true">  
    <column name="EMPNO"/>  
  </field>  
  ...  
</class>
```

package.jdo

Tożsamość nietrwała

- Nadanie klasie tożsamości nietrwałej (nondurable identity) powoduje brak jednoznacznego identyfikatora w składnicy danych. Mechanizm stosuje się dla danych, dla których nigdy nie zachodzi dostęp na podstawie wartości klucza podstawowego (np. pozycje w pliku dziennika).

```
<class name="Logger" identity-type="nondurable">  
...  
</class>
```

package.jdo

Odwzorowanie obiektowo-relacyjne

Odwzorowanie O/R

- Odwzorowanie pojedynczej klasy na pojedynczą tabelę

```
package pl.org.ploug.szkola2006.jdo;

public class Department {
    private int deptNo;
    private String name;
    private String location;
    ...
}
```

Department

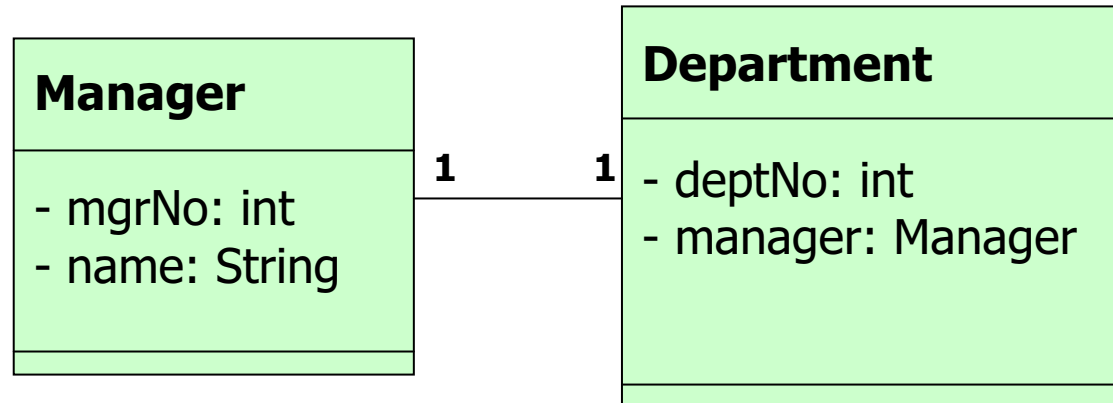
- deptNo: int
- name: String
- location: String

package.jdo

```
<class name="Department" identity-type="application" table="DEPT">
  <field name="deptNo" primary-key="true">
    <column name="DEPTNO" precision="2" allows-null="false">/>
  </field>
  <field name="name">
    <column name="DNAME" length="14" jdbc-type="VARCHAR">/>
  </field>
  <field name="location">
    <column name="LOC" length="13" jdbc-type="VARCHAR" default-value="NYC">/>
  </field>
</class>
```

Odwzorowanie O/R

- Związek 1-1
 - jednokierunkowy



```

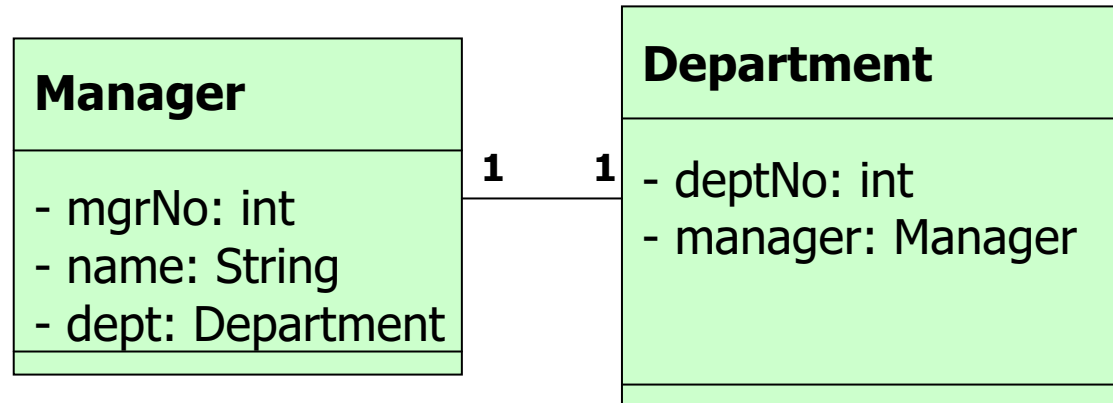
<class name="Department" identity-type="application" table="DEPT">
  <field name="deptNo" primary-key="true" persistence-modifier="persistent">
    <column name="DEPTNO" precision="2" jdbc-type="NUMBER"/>
  </field>
  <field name="manager" persistence-modifier="persistent">
    <column name="MGR" precision="10" jdbc-type="NUMBER"/>
  </field>
</class>
<class name="Manager" identity-type="application" table="EMP">
  <field name="mgrNo" primary-key="true" persistence-modifier="persistent">
    <column name="EMPNO" precision="4" jdbc-type="NUMBER">
  </field>
  <field name="name" persistence-modifier="persistent">
    <column name="ENAME" length="10" jdbc-type="VARCHAR"/>
  </field>
</class>

```

package.jdo

Odwzorowanie O/R

- Związek 1-1
 - dwukierunkowy
 - jeden klucz obcy
 - dwa klucze obce



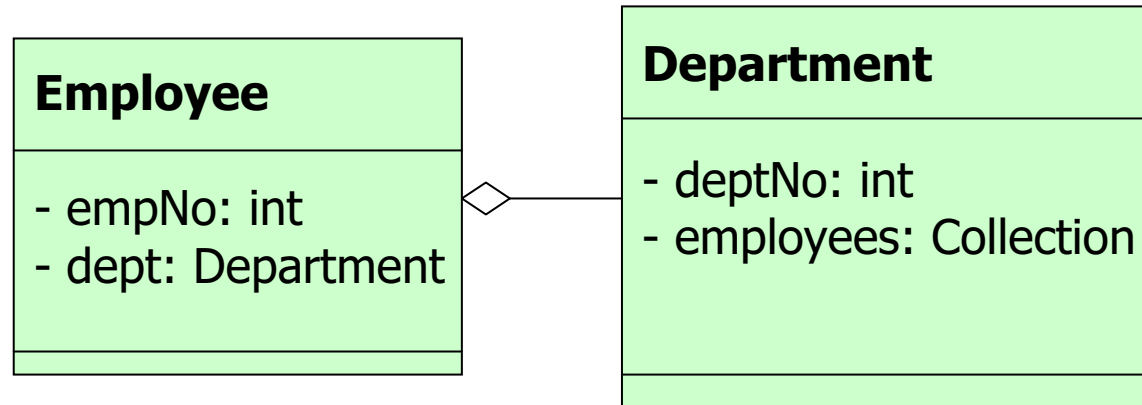
```

<class name="Department" identity-type="application" table="DEPT">
  <field name="deptNo" primary-key="true" persistence-modifier="persistent">
    <column name="DEPTNO" precision="2" jdbc-type="NUMBER"/>
  </field>
  <field name="manager" persistence-modifier="persistent">
    <column name="MGR" precision="10" jdbc-type="NUMBER"/>
  </field>
</class>
<class name="Manager" identity-type="application" table="EMP">
  ...
  <field name="dept" persistence-modifier="persistent" mapped-by="user">
    <column name="DEPTNO" precision="2" jdbc-type="NUMBER"/>
  </field>
</class>
  
```

package.jdo

Odwzorowanie O/R

- Związek 1-N
 - List, Set, Map
 - tabela, klucz obcy
 - jedno- i dwukierunkowy



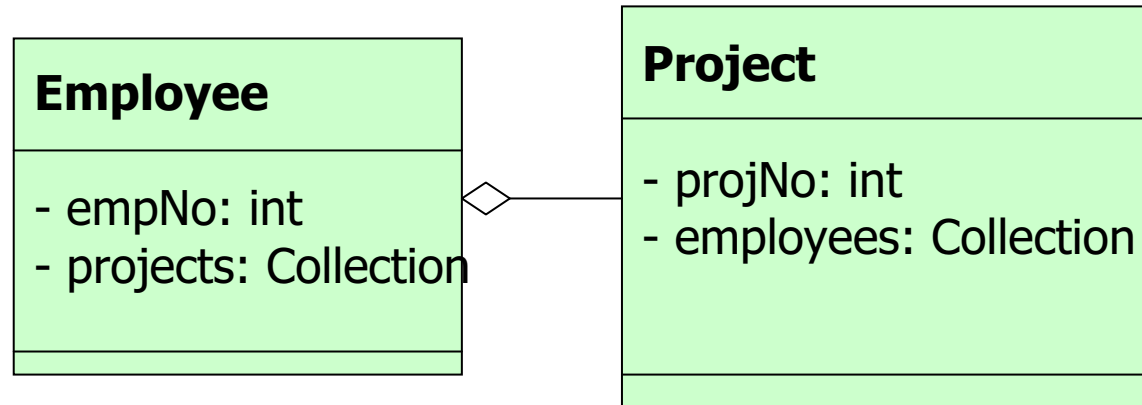
```

<class name="Department" identity-type="application" table="DEPT">
  ...
  <field name="employees" persistence-modifier="persistent"
    mapped-by="department">
    <collection element-type="pl.org.ploug.szkola2006.jdo.Employee"/>
  </field>
</class>
<class name="Employee" identity-type="application" table="EMP">
  ...
  <field name="dept" persistence-modifier="persistent" mapped-by="user">
    <column name="DEPTNO" precision="2" jdbc-type="NUMBER"/>
  </field>
</class>
  
```

package.jdo

Odwzorowanie O/R

- Związek N-M
 - List, Set, Map
 - tabela, dwie tabele



```

<class name="Project" identity-type="application" table="PROJ">
  ...
  <field name="employees" persistence-modifier="persistent"
    mapped-by="projects">
    <collection element-type="pl.org.ploug.szkola2006.jdo.Employee"/>
  </field>
</class>
<class name="Employee" identity-type="application" table="EMP">
  ...
  <field name="projects" persistence-modifier="persistent" table="EMP_PROJ">
    <collection element-type="pl.org.ploug.szkola2006.jdo.Project"/>
    <join><column name="EMPNO"/></join>
    <element><column name="PROJNO"/></element>
  </field>
</class>

```

package.jdo

Dodatkowe możliwości O/R

- JDO umożliwia także następujące operacje:
 - utrwalanie obiektów zagnieżdżonych,
 - serializację składowych do składnicy danych (dotyczy przede wszystkim składowych z interfejsu `Collections`),
 - definiowanie indeksów na polach,
 - definiowanie ograniczeń unikalnych
 - definiowanie ograniczeń referencyjnych i integralnościowych,
 - utrwalanie hierarchii klas,
 - utrwalanie składowych będących tablicami
 - w postaci osobnej kolumny,
 - w postaci serializowanej,
 - przy użyciu tabeli połączeniowej,
 - przy wykorzystaniu klucza obcego.

Transakcje i zapytania

Transakcje

- Zarządzanie transakcjami
 - lokalne: użytkownik jawnie podaje początek i koniec transakcji,
 - przez szkielet aplikacyjny: transakcje są rozpoczynane i kończone automatycznie przez szkielet (np. Spring),
 - przez kontener J2EE: kontrola nad transakcjami jest całkowicie przekazana do kontenera.
- Typy transakcji
 - pesymistyczne: przed wykonaniem operacji na obiekcie zakładana jest blokada, możliwe stosowanie blokad intencjonalnych (SELECT ... FOR UPDATE ...), możliwe ustalanie poziomów izolacji transakcji,
 - optymistyczne: przed wykonaniem operacji sprawdzana jest spójność danych na podstawie zawartości znacznika czasowego
 - operacje wykonywane poza transakcjami
 - `javax.jdo.option.NontransactionalRead`
 - `javax.jdo.option.NontransactionalWrite`

Transakcje

```
PersistenceManagerFactory pmf = JDOHelper.getPersistenceManagerFactory(props);
PersistenceManager pm = pmf.getPersistenceManager();

Transaction tx = pm.currentTransaction();
try {
    tx.begin();

    // kod aplikacji
    tx.setSynchronization(new javax.transaction.Synchronization() {
        public void beforeCompletion() { // przed zatwierdzeniem lub wycofaniem }
        public void afterCompletion(int status) {
            if (status == javax.transaction.Status.STATUS_ROLLEDBACK)
                // transakcja została wycofana
            if (status == javax.transaction.Status.STATUS_COMMITTED)
                // transakcja została zatwierdzona
        } });

    tx.commit();
}
finally
{
    if (tx.isActive()) {
        tx.rollback();
    }
}
```

Zapytania

- Zapytanie wykorzystujące rozszerzenie klasy

```
Extent e = pm.getExtent(Department.class,true);
Query q = pm.newQuery(e);
q.setOrdering("name ascending"); ← porządek krotek
Collection c = (Collection)q.execute(); ← wykonanie zapytania
Iterator i = c.iterator();

while (i.hasNext()){
    Department d = (Department)i.next();
    System.out.println("Department " + d.getName() + " " + d.getLocation());
}
```

```
Department ACCOUNTING  NEW YORK
Department CONTROLLING NEW YORK
Department OPERATIONS  BOSTON
Department RESEARCH    DALLAS
Department SALES        CHICAGO
```

Zapytania

- Zapytanie nazwane w pliku konfiguracyjnym

```
Query q = pm.newNamedQuery(Department.class, "NewYorkDepartments");  
Collection c = (Collection)q.execute();  
Iterator i = c.iterator();  
  
while (i.hasNext()){  
    Department d = (Department)i.next();  
    System.out.println("Department " + d.getName() + " " + d.getLocation());  
}
```

```
Department ACCOUNTING NEW YORK  
Department CONTROLLING NEW YORK
```

```
<class>  
...  
    <query name="NewYorkDepartments" language="javax.jdo.query.JDOQL">  
        <![CDATA[ SELECT FROM Department WHERE location == "NEW YORK" ]]>  
    </query>  
</class>
```

package.jdo

Zapytania

- Zapytanie JDOQL

```
String filter = " location == pLocation && employees.contains(emp) " +
    " && emp.job == pJob && emp.name.startsWith(\"S\") && emp.salary > 1000";
```

```
Extent e = pm.getExtent(Department.class,true);
```

```
Query q = pm.newQuery(e,filter);
```

```
q.declareImports("import pl.org.ploug.szkola2006.jdo.Employee");
```

```
q.declareVariables("Employee emp");
```

```
q.declareParameters("String pLocation, String pJob");
```

zmienne

zmienne wiązania

```
Collection c = (Collection)q.execute("DALLAS","ANALYST");
```

```
Iterator i = c.iterator();
```

```
while (i.hasNext()){
```

```
    Department d = (Department)i.next();
```

```
    System.out.println("Department " + d.getName() + " " + d.getLocation());
```

```
}
```

```
Department RESEARCH DALLAS
```

Zapytania

- Zapytanie SQL

```
Query q = pm.newQuery("javax.jdo.query.SQL",
    "SELECT dname, count(ename)
    FROM dept LEFT OUTER JOIN emp USING (deptno)
    GROUP BY dname");

List l = (List)q.execute();
Iterator i = l.iterator();

while (i.hasNext()){
    Object[] o = (Object[])i.next();
    System.out.println("Department " + o[0] + ", number of employees: " + o[1]);
}
```

```
Department ACCOUNTING, number of employees: 3
Department OPERATIONS, number of employees: 0
Department CONTROLLING, number of employees: 1
Department RESEARCH, number of employees: 5
Department SALES, number of employees: 6
```

Zapytania

- Zapytanie złożone

```
Extent e = pm.getExtent(Employee.class, true);
Query q = pm.newQuery(e);

q.setOrdering("name ascending");
q.setRange(1,5); ← ograniczenie rozmiaru wyniku
Collection c = (Collection)q.execute();
Iterator i = c.iterator();

while (i.hasNext()){
    Employee emp = (Employee)i.next();
    String boss = ((emp.getBoss() != null)) ? emp.getBoss().getName() : "";
    String department = ((emp.getDepartment() != null)) ?
        emp.getDepartment().getName() : "";
    System.out.println("Employee " + emp.getName() + " Department: " + department
        + " Boss: " + boss);
}
```

```
Employee ALLEN Department: SALES Boss: BLAKE
Employee BLAKE Department: SALES Boss: KING
Employee CLARK Department: ACCOUNTING Boss: KING
Employee FORD Department: RESEARCH Boss: JONES
```

Metody callback

- Trwałe klasy mogą implementować interfejs **InstanceCallbacks** informujący klasę o zdarzeniach związanych z cyklem życia danej klasy.

```
public class Employee implements InstanceCallbacks {
    String name;
    String job;
    ...
    public void jdoPostLoad() { ... }
    public void jdoPreClear() { ... }
    public void jdoPreStore() { ... }
    public void jdoPreDelete() { ... }
}

public interface AttachCallback
{
    public void jdoPreAttach();
    public void jdoPostAttach(Object attached);
}

public interface DetachCallback
{
    public void jdoPreDetach();
    public void jdoPostDetach(Object detached);
}
```

Zarządzanie pamięcią cache i dziennikiem

- Poziomy pamięci cache wymagane przez JDO
 - poziom 1: instancje buforowane w `PersistentManager`
 - **WeakRefCache, SoftRefCache, HardRefCache**
 - poziom 2: instancje buforowane w `PersistentManagerFactory`
- JDO wykorzystuje bibliotekę Log4J (<http://jakarta.apache.org/log4j>) do zapisu plików dziennika. Biblioteka ta umożliwia elastyczną konfigurację dziennika
 - kategorie zdarzeń,
 - poziomy komunikatów,
 - format pliku dziennika.