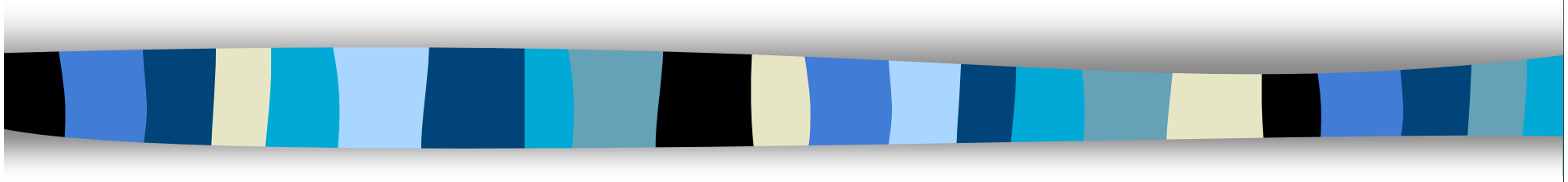
Frameworks de Persistencia en Java



Rubén Inoto Soto

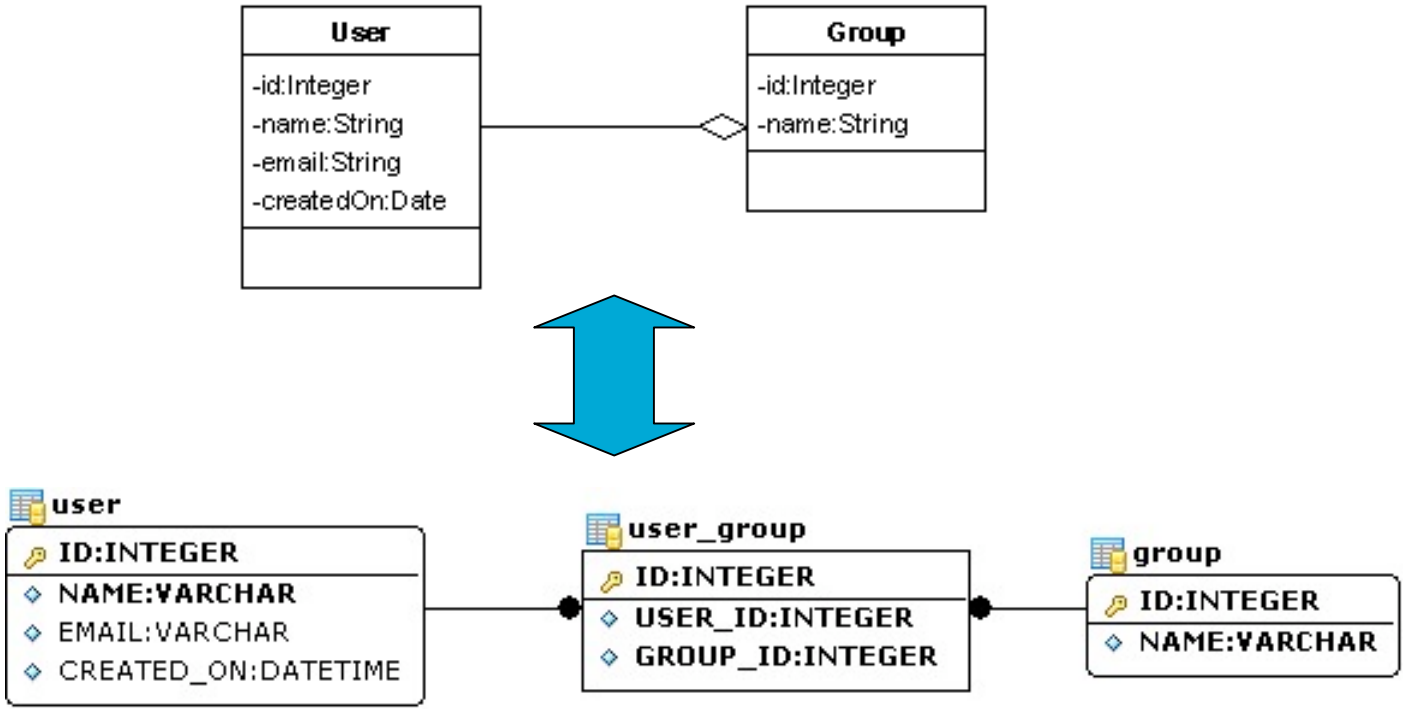
Universidad de Alicante – 12 de Mayo, 2006

Persistencia

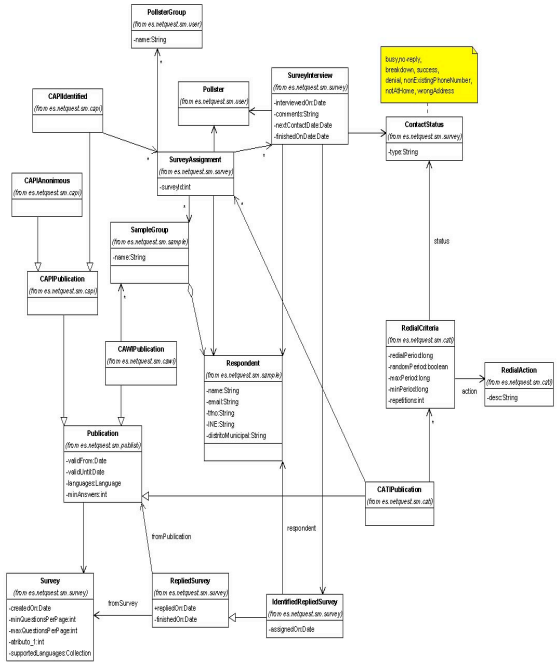
- Del lat. persistere → Durar por largo tiempo
- Almacenamiento de la información después de la finalización del programa
- La mayoría de los programas actuales necesitan preservar los datos para posterior uso
- El sistema más común se basa en bases de datos relacionales

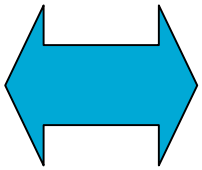
Problema

- Mapeo de Objetos (estrucutra jerárquica) a Base de Datos Relacional (estructura tabular) y viceversa

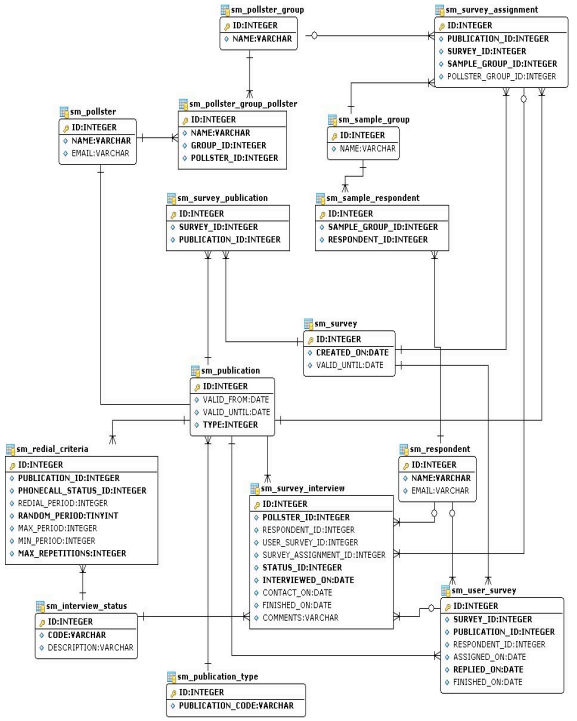


Problema Cont.





- Asociaciones
- Herencia
- Polimorfismo
- Cascada
- Cache
- Lazy Loading
- Paginación
- ...



JDBC - Características

- Interfaz de programación para acceso a BD
- Rutinas para manipular los datos y metadatos
- Acceso „directo“ a la base de datos
- Control de las conexiones y transacciones (autoCommit)
- Lenguaje SQL

JDBC – Ejemplo 1

- Cargar el Driver y Obtener Conexión a BD

```
Class.forName("com.mysql.jdbc.Driver");
Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/persistenceTest", "root",
„xxx");
```

- Transacciones y Sesiones

```
con.setAutoCommit(false);
// Codigo...
con.commit();
```

Código poco intuitivo

- Ejecutar Query

Acceso directo a SQL

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT NAME as name, EMAIL as email " +
                        "FROM UA_USER where ID = " + 1);
User user = new User();
if ( rs.next() ) {
   user.setName(rs.getString("name"));
   user.setEmail(rs.getString("email"));
}
rs.close();
stmt.close();
```

Mapping Manual

JDBC – Ejemplo 1. Cont

- PreparedStatement

```
PreparedStatement createUserStmt =
  con.prepareStatement("INSERT INTO UA_USER (NAME, EMAIL) VALUES (?, ?)");
createUserStmt.setString(1, "ruben");
createUserStmt.setString(2, "ruben@ruben.com");

createUserStmt.execute();
```

- Cerrar conexiones

```
con.close();
```

JDBC - Problemas

- Código específico de Base de Datos
  - mySQL: outer join, autoincrement
  - Oracle (+), Sequences
- Acceso directo a conexiones y transacciones
- Transacciones a un único DataSource
- El desarrollador Java debe saber tambien otro lenguaje (SQL)
- El código se hace repetitivo
  - Inserts, Selects, Updates, Deletes para cada objeto del modelo

iBATIS

- Nace en 2001
- Compuesto de dos partes
  - Capa DAO
    - Capa general que abstrae el acceso a datos
  - SQLMaps
    - Envuelve las llamadas a JDBC, definidas en un XML
    - Mapeo entre objetos (JavaBeans) y SQL (Statements o StoredProcedures)
- No es un ORM puro
  - Aún tenemos que escirbir los SQL
  - El mapping se realiza a partir de los resultados del SQL

iBATIS – Ejemplo Java

- Obtener SQL Map

```
Reader reader = Resources.getResourceAsReader("sqlMapConfig.xml");
sqlMap = SqlMapClientBuilder.buildSqlMapClient(reader);
```

- Transacciones y Sesiones

```
sqlMap.startTransaction();
// Codigo...
sqlMap.commitTransaction();
sqlMap.endTransaction ();
```

- Ejecutar Query

```
Integer userId = (Integer) sqlMap.insert("insertUser", user);
```

```
User user = (User) sqlMap.queryForObject("getUser", userId);
```

```
sqlMap.update("updateUser", userToUpdate);
```

iBATIS – Ejemplo XML

- SQL Map: Configuración general

```
<sqlMapConfig>

        <settings cacheModelsEnabled="true"
                  enhancementEnabled="true"
                  lazyLoadingEnabled="true"
                  maxRequests="32"
                  maxSessions="10"
                  maxTransactions="5"
                  useStatementNamespaces="false" />

      <transactionManager type="JDBC">
        <dataSource type="SIMPLE">
          <property name="JDBC.Driver" value="com.mysql.jdbc.Driver" />
                ...
        </dataSource>
      </transactionManager>

      <sqlMap resource="user.xml" />
      <sqlMap resource="group.xml" />

    </sqlMapConfig>
```

Pluggable DataSources

iBATIS – Ejemplo XML

- Mapeo de Objetos

```
<sqlMap namespace="User">
      <cacheModel id="userCache" type="LRU">
                  <flushInterval hours="24" />
                  <flushOnExecute statement="insertUser" />
                  <flushOnExecute statement="updateUser" />
                  <flushOnExecute statement="deleteUser" />
                  <property name="size" value="1000" />
            </cacheModel>
   <select id="getUser" resultClass="es.ua.ibatis.User">
     SELECT ID as id, NAME as name, EMAIL as email, CREATED_ON as createdOn
     FROM UA_USER WHERE ID = #value#
    </select>

    <insert id="insertUser" parameterClass="es.ua.ibatis.User">
       INSERT INTO UA_USER (ID, NAME, EMAIL, CREATED_ON)
       VALUES (#id#, #name#, #email#, SYSDATE())
        <selectKey keyProperty="id" resultClass="int">
           SELECT LAST_INSERT_ID() AS value
        </selectKey>
    </insert>
    <update id="updateUser" parameterClass="es.ua.ibatis.User">
             UPDATE UA_USER
            SET NAME = #name#, EMAIL = #email# WHERE ID = #id#
    </update>
    <delete id="deleteUser" parameterClass="es.ua.ibatis.User">
           DELETE UA_USER WHERE ID = #id#
    </delete>
    </sqlMap>
```

Cache intuitiva

SQL directo

Código específico de proveedor (mySQL)

iBATIS

- Ventajas
  - Simplicidad (Fácil de usar, si sabemos SQL)
  - Curva de aprendizaje (muy intuitivo al empezar a usar)
  - Abstrae (en cierto modo) JDBC
  - Control completo sobre los SQL
  - Rendimiento: cache fácilmente configurable (LRU, FIFO, Memory, OSCache)
  - Flexibilidad (statements dinámicos)

```
select * from ACCOUNT
  <isGreaterThan prepend="and" property="id" compareValue="0">
     where ACC_ID = #id#
  </isGreaterThan>
order by ACC_LAST_NAME
```

  - Permite mapeo directo a XML
  - Batches, Lazy Loading, Transacciones, Paginación

iBATIS

- Desventajas
  - No es un ORM
  - El desarrollador debe tener altos conocimientos de SQL
  - Transacciones restringidas a un DataSource
  - No es independiente del proveedor de BD (e.g. secuencias): baja portabilidad

```
<sqlMap namespace="Group">
        <insert id="insertGroup„ parameterClass="es.ua.ibatis.Group">
          INSERT INTO UA_GROUP (ID, NAME) VALUES (#id#, #name#)
           <selectKey keyProperty="id" resultClass="int">
             SELECT LAST_INSERT_ID() AS value
          </selectKey>
      </insert>
 </sqlMap>
```

ORM (Object Relational Mapping)

- Realizan el mapeo de objetos a tablas y viceversa
- Aceleran el trabajo
  - El desarollador se concentra en el Modelo de Objetos, y deja el mapping a la herramienta
- Se encargan de gestionar asociaciones, herencia, poliformismo, lazy loading..
- También suelen proporcionar servicios para facilitar el manejo de conexiones y transacciones

EJB (<= 2.x)

- Especificación – 1.1 → 1999, 2.1 → 2003
- No es sólo un framework de persistencia
  - Framework para el desarrollo y deployment de aplicaciones distribuidas.
- Entra fuerte con gran soporte por parte de los proveedores
  - Forman la especificación BEA, IBM, Oracle, Sybase, SUN..
- Escalabilidad, Soporte de Transacciones, Seguridad, Concurrencia
- Define 2 tipos de persistencia
  - BMP: La aplicación debe persistir los datos explícitamente
  - CMP: El contenedor es el encargado de gestionar la persistencia de los objetos

EJB (<= 2.x) - Objetivos

- Convertirse en la arquitectura standard para construir aplicaciones OO en Java
- Liberar al desarrollador de „detalles“ como persistencia, transacciones, seguridad , conexiones, pooling, multi-thread, etc..
- Construir aplicaciones a partir de módulos independientes desarrollados por diferentes proveedores
- Independencia de plataforma
- Pretende ser la solución a todos los problemas

EJB (<= 2.x) – Ejemplo: Clases

Demasiadas Clases

- Bean Interface (User)
  - Interfaz que extiende EJBObject
  - Representa el Objeto a persistir
- Bean Home (UserHome)
  - Interfaz que extiende EJBHome
  - Contiene metodos para crear el bean, y los finders
- Clase Bean (UserBean)
  - Implementa EntityBean (ejbXXX), y los métodos de nuestra BeanInterface

EJB (<= 2.x) – Deployment Descriptors

- ejb-jar.xml

```
<ejb-jar >
    <!-- Entity Beans -->
    <entity >
      <description><![CDATA[User Bean]]></description>

      <ejb-name>Use</ejb-name>

      <local-home>ua.es.ejb.UserLocalHome</local-home>
      <local>ua.es.ejb.UserLocal</local>

      <ejb-class>ua.es.ejb.UserCMP</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.Integer</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>User</abstract-schema-name>
      <cmp-field >
        <description><![CDATA[gets id (primary key)]]></description>
        <field-name>id</field-name>
      </cmp-field>
      ...
      <query>
        <query-method>
          <method-name>findById</method-name>
          <method-params>
            <method-param>java.lang.Integer</method-param>
          </method-params>
        </query-method>
        <result-type-mapping>Local</result-type-mapping>
        <ejb-ql><![CDATA[SELECT DISTINCT OBJECT(c) FROM User AS c WHERE c.id = ?1]]></ejb-ql>
      </query>
    </entity>

    <method-permission >
     ....
    </method-permission>

    <container-transaction >
      <method >
        <ejb-name>User</ejb-name>
         <method-name>*</method-name>
      </method>
      <trans-attribute>Mandatory</trans-attribute>
    </container-transaction>

</ejb-jar>
```

Excesivamente complejo

EJB (<= 2.x) – Deployment Descriptors

- weblogic-ejb-jar.xml

```
<weblogic-ejb-jar>
 <description><![CDATA[Generated by XDoclet]]></description>
  <weblogic-enterprise-bean>
    <ejb-name>User</ejb-name>
    <entity-descriptor>
      <persistence>
        <persistence-use>
          <type-identifier>WebLogic_CMP_RDBMS</type-identifier>
          <type-version>7.0</type-version>
          <type-storage>META-INF/weblogic-cmp-rdbms-jar.xml</type-storage>
        </persistence-use>
      </persistence>
    </entity-descriptor>
    <reference-descriptor>
    </reference-descriptor>

    <local-jndi-name>ejb/ua/ejb/UserLocal</local-jndi-name>
  </weblogic-enterprise-bean>

</weblogic-ejb-jar>
```

Depende del proveedor

EJB (<= 2.x) – Deployment Descriptors

- weblogic-cmp-rdbms-jar.xml

```
<weblogic-rdbms-jar>
  <weblogic-rdbms-bean>
    <ejb-name>User</ejb-name>
    <data-source-name>UADatasource</data-source-name>
    <table-map>
    <table-name>UA_USER</table-name>
    <field-map>
      <cmp-field>id</cmp-field>
      <dbms-column>ID</dbms-column>
    </field-map>
    <field-map>
      <cmp-field>name</cmp-field>
      <dbms-column>NAME</dbms-column>
    </field-map>
    </table-map>

    <weblogic-query>
      <query-method>
        <method-name>findById</method-name>
        <method-params>
          <method-param>java.lang.Integer</method-param>
        </method-params>
      </query-method>
      <weblogic-ql><![CDATA[SELECT DISTINCT OBJECT(c) FROM User AS c WHERE c.id = ?1]]></weblogic-ql>
    </weblogic-query>

</weblogic-rdbms-jar>
```

Depende del proveedor

EJB (<= 2.x) – Ejemplo con Xdoclet

```
/**
 * @ejb.bean
 *      type="CMP"
 *      cmp-version="2.x"
 *      name="User"
 *      schema="User"
 *      jndi-name="ejb/ua/user/User"
 *      local-jndi-name="ejb/ua/user/UserLocal"
 *      view-type="local"
 *      primkey-field="id"
 *      reentrant="False"
 *
 * @ejb.transaction type="Mandatory"
 * @ejb.persistence table-name="UA_USER"
 * @ejb.pk class="java.lang.Integer"
 * @ejb.home generate="local"
 *
 * @ejb.interface generate="local"
 *
 * @ejb.finder
 *      signature="java.util.Collection findByName()"
 *      unchecked="true"
 *      query="SELECT OBJECT(user) FROM User user where user.name = ?1"
 *      result-type-mapping="Local"
 *
 * @ejb.value-object name="User" match="*"
 *
 * @weblogic.automatic-key-generation
 *      generator-type="ORACLE"
 *      generator-name="UA_USER_SQ"
 *      key-cache-size="1"
 *
 * @ejb.util generate="physical"
 *
 * @weblogic.cache
 *      max-beans-in-cache="1000"
 *      idle-timeout-seconds="600"
 *      concurrency-strategy ="Database"
 */
public abstract class UserBean implements EntityBean {
```

Sigue siendo complejo

EJB (<= 2.x) – Ejemplo con Xdoclet Cont

```
public abstract class UserBean implements EntityBean {
    /**
     * @ejb.create-method
     */
    public Integer ejbCreate(UserValue userValue)
            throws CreateException {
        return null;
    }
    public void ejbPostCreate(UserValue userValue)
            throws CreateException {
        setUserValue(userValue);
    }

    /**
     * @ejb.pk-field
     * @ejb.persistent-field
     * @ejb.interface-method
     *       view-type="local"
     * @ejb.persistence
     *       column-name="ID"
     */
    public abstract Integer getId();
    public abstract void setId(Integer id);


    /**
     * @return     created user account ID of the userPlaylist object
     *
     * @ejb.interface-method view-type="local"
     * @ejb.persistence column-name="NAME"
     */
    public abstract Integer getName();
    public abstract void setName(Integer name);

    /**
     * @ejb.interface-method view-type="local"
     */
    public abstract UserValue getUserValue();
    public abstract void setUserPlaylistValue(UserPlaylistValue value);
}
```

Sigue siendo complejo

EJB (<= 2.x) – Ejemplo Asociación

```
/**
     * @ejb.interface-method view-type="local"
     *
     * @ejb.relation
     *      name="group-groupUser"
     *      role-name="group-has-users"
     *      target-ejb="UserGroup"
     *      target-role-name="userGroup-in-group"
     *      target-cascade-delete="no"
     *
     * @ejb.value-object
     *      compose="UserGroupValue"
     *      compose-name="UserGroup"
     *      members="UserGroupLocal"
     *      members-name="UserGroup"
     *      relation="external"
     *      type="java.util.Collection"
     *
     * @weblogic.target-column-map
     *      foreign-key-column="GROUP_ID"
     */
    public abstract Collection getGroupUsers();
    public abstract void setGroupUsers(Collection user
```

Group

UserGroup

```
/**
     * @ejb.relation
     *      name="UserGroup-User"
     *      role-name="UserGroup-referencesOne-User"
     *      cascade-delete="no"
     *      target-ejb="User"
     *      target-role-name="User-isOnIndex-UserGroup"
     *      target-multiple="yes"
     *
     * @weblogic.column-map
     *      foreign-key-column="USER_ID"
     *
     * @ejb.value-object
     *      aggregate="ua.es.ejv.UserValue"
     *      aggregate-name="UserValue"
     *      members="ua.es.ejb.UserLocal"
     *      members-name="User"
     *      relation="external"
     */
    public abstract UserLocal getUser();
```

EJB (<= 2.x) - Acceso a Bean Local

- InitialContext y LocalHome

```
InitialContext initialContext = new InitialContext();
UserLocalHome userLocalHome = (UserLocalHome) initialContext.lookup(jndiName);
```

Necesitamos un contenedor de aplicaciones

- Crear usuario

```
userLocalHome.create(userValue);
```

- Update

```
UserLocal userLocal = userLocalHome.findById(userId);
userLocal.setUserValue(newUserValue);
```

Obtenemos un objeto „Local“, no nuestro POJO.
Necesitamos mapearlo

EJB (<= 2.x) - Problemas

- **Curva de aprendizaje**
- Necesita un contenedor de aplicaciones
- **Intrusivo**: las clases a persistir deben implementar interfaces EJB
- Excesivos deployment descriptors
  - Herramientas como Xdoclet „intentan ayudar“
- Acceso a las propiedades de un objeto siempre son tratadas como remotas, por lo que el rendimiento es muy bajo
  - EJB 2 soluciona este problema creando las interfaces „Locales“
- Se hace necesario introducir ValueObjects Pattern
- No soporta Herencia ni relaciones entre clases!
  - EJB 2 introduce CMR para asociaciones
- Conseguir un buen rendimiento no es fácil
- A pesar de ser un estandard, cada proveedor tiene demasiadas extensiones (portar la aplicación no es tan sencillo)
- **Testing** se hace muy difícil y lento
  - Cactus ayuda, pero sigue siendo lento

EJB (<= 2.x) - Ventajas

- Provee servicios de forma transparente
  - Transacciones, Seguridad, Conexiones a BD
- Rendimiento, si se llega a dominar (cache)
- Clustering
- RMI
- Muchos servidores de aplicaciones importantes lo soportan

JDO (<= 1.x)

- Especificación - JSR 12 (V1.0 → 2002)
- Nace al no existir una interfaz estandar en Java para la persistencia de atos
- Objetivos
  - Interfaz **estandar** del sistema de almacenamiento de datos (RDB, FS, OODB, XMLDB..)
  - Persistir datos de forma transparente en cualquier dispositivo (no sólo Bases de Datos Relacionales)
- POJOs
  - Plain Old Java Object
  - Objeto Java simple
  - No implementan interfaces del framework
  - No intrusión

JDO (<= 1.x) - Características

- Portabilidad
  - Al ser una interfaz, puede cambiarse la implementación sin cambiar el código
- Independencia de la Fuente de Datos
- Rendimiento (dependiendo de la implementación)
- Con o sin container (con container provee de transacción, seguridad, gestion de conexiones)
- Byte Code Enhancement vs Reflection: Transforma la clase anyadiendo el código necesario para permitir la sincronización de ésta con su representación en la base de datos

JDO (<= 1.x)– Ejemplo XML

- Mapeo XML

Extensiones de proveedor si queremos especificar el nombre de tablas y columnas

```
<jdo>
        <package name="es.ua.jdo">
         <class name="User" identity-type="datastore" requires-extent="true">
          <extension vendor-name="jpox" key="table-name" value="UA_USER" />
          <extension vendor-name="jpox" key="key-column-name" value="ID" />
          <field name="id" primary-key="true" null-value="none">
              <extension vendor-name="jpox" key="column-name" value="ID" />
          </field>
          <field name="name" persistence-modifier="persistent" primary-key="false">
              <extension vendor-name="jpox" key="length" value="max 100" />
          </field>
          <field name="email" persistence-modifier="persistent" primary-key="false">
              <extension vendor-name="jpox" key="length" value="max 255" />
          </field>
         </class>
         <class name="Group" identity-type="datastore" requires-extent="true">
            <extension vendor-name="jpox" key="table-name" value="UA_GROUP" />
            <extension vendor-name="jpox" key="key-column-name" value="ID" />
             <field name="id" primary-key="true" null-value="none">
                <extension vendor-name="jpox" key="column-name" value="ID" />
            </field>
            <field name="name" persistence-modifier="persistent" primary-key="false" >
                <extension vendor-name="jpox" key="length" value="max 100" />
            </field>
             <field name="users" persistence-modifier="persistent" primary-key="false">
              <collection element-type="es.ua.jdo.User">
                <extension vendor-name="jpox" key="table-name" value="UA_USER_GROUP" />
                <extension vendor-name="jpox" key="owner-column-name" value="GROUP_ID" />
                <extension vendor-name="jpox" key="element-column-name" value="USER_ID" />
              </collection>
        </field>
       </class>
      </package>
     </jdo>
```

Asociación 1:N

JDO (<= 1.x) – Ejemplo Java

- Persistence Manager

```
Properties properties = new Properties();
properties.setProperty("javax.jdo.PersistenceManagerFactoryClass",
"org.jpox.PersistenceManagerFactoryImpl");
properties.setProperty("javax.jdo.option.ConnectionDriverName","com.mysql.jdbc.Driver");
...

PersistenceManagerFactory pmfactory = JDOHelper.getPersistenceManagerFactory(properties);
PersistenceManager persistenceMgr = pmfactory.getPersistenceManager();
```

- Transacciones

```
Transaction transaction = persistenceMgr.currentTransaction();
transaction.begin();
//Codigo
transaction.commit();
persistenceMgr.close();
```

JDO (<= 1.x)– Ejemplo Java

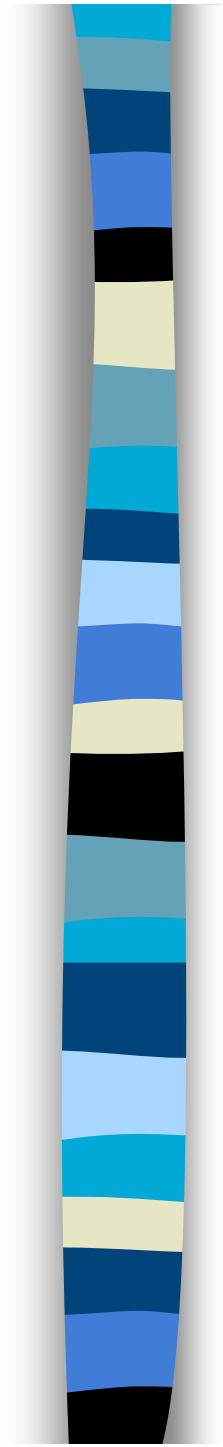
- Manipulando Objetos

```
User product = new User("ruben", "ruben@ruben.com");
persistenceMgr.makePersistent(product);
```

```
Extent extent = persistenceMgr.getExtent(User.class, true);
Query query = persistenceMgr.newQuery(extent, "name = ruben");
query.setOrdering("name ascending");
Collection users = (Collection) query.execute();
```

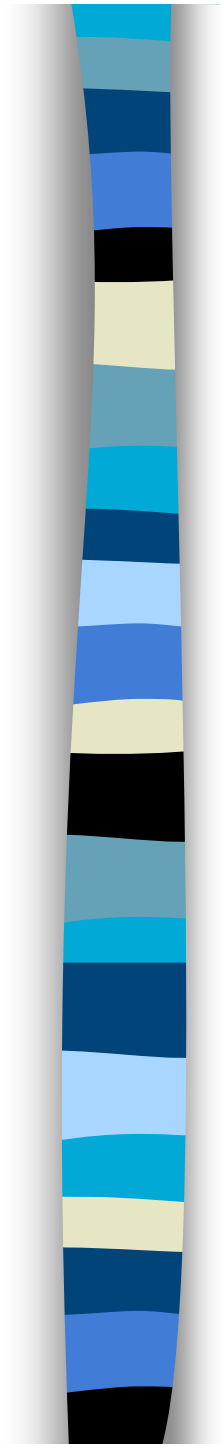
JDO (<= 1.x)

- Ventajas
  - „estandard“
  - Rendimiento (diferentes implementaciones)
    - Sun reference Implementation
    - Kodo (comercial), ahora BEA
    - Apache OJB 1.0.4
    - JPox (muy activa, RI para JDO2)
- Desventajas
  - Poco peso político (en comparación con EJB)
  - Code enhancement (paso adicional a la compilación)
  - XML mapping (herramientas como XDoclet pueden ayudar)



Hibernate

- Creada por Gavin King a finales del 2001 como alternativa a EJB CMP
- Se una a JBoss finales del 2003
  - Provee training (pagando)
- Pretende ser la solución completa para el problema de persistencia en Java
- No intrusivo
- Muy buena documentación (forums para ayuda, libro)
- Comunidad activa con muchos usuarios



Hibernate - Características

- No intrusivo (estilo POJO)
- Muy buena documentación (forums para ayuda, libro)
- Comunidad activa con muchos usuarios
- Transacciones, caché, asociaciones, poliformismo, herencia, lazy loading, persistencia transitiva, estrategias de fetching..
- Potente lenguaje de Consulta (HQL)
  - subqueries, outer joins, ordering, proyeccion (report query), paginación)
- Fácil testeo
- No es estandard

Hibernate - Ejemplo

- Configuración principal

```
<sqlMap namespace="User">
        <select id="getUser" resultClass="es.ua.ibatis.User">
          SELECT ID as id, NAME as name, EMAIL as email, CREATED_ON as createdOn
          FROM UA_USER WHERE ID = #value#
        </select>

        <insert id="insertUser" parameterClass="es.ua.ibatis.User">
         INSERT INTO UA_USER (ID, NAME, EMAIL, CREATED_ON)
         VALUES (#id#, #name#, #email#, SYSDATE())
            <selectKey keyProperty="id" resultClass="int">
             SELECT LAST_INSERT_ID() AS value
          </selectKey>
        </insert>

          <update id="updateUser" parameterClass="es.ua.ibatis.User">
           UPDATE UA_USER
          SET NAME = #name#, EMAIL = #email# WHERE ID = #id#
        </update>

         <delete id="deleteUser" parameterClass="es.ua.ibatis.User">
          DELETE UA_USER WHERE ID = #id#
        </delete>
    </sqlMap>
```

Hibernate – Ejemplo Java

- Transacciones y sesiones

```
Session session = getSessionFactory().openSession();
Transaction tx = session.beginTransaction();

    // Codigo...

tx.commit();
session.close();
```

- Clases persistentes

```
User user = new User("ruben", "ruben@ruben.com");
Integer userId = (Integer) session.save(user);
User userGot = (User) session.get(User.class, userId);
```

Hibernate – Ejemplo Java

- Asociacion Grupo → Usuario

```
<set name="users" table="UA_USER_GROUP">
        <key column="GROUP_ID" on-delete="noaction" />
        <many-to-many column="USER_ID"
                class="es.ua.hibernate.User" />
</set>
```

```
User user = (User) session.get(User.class, 1);

Group group = new Group();
group.setName("group");
group.addUser(user);

Integer groupId = (Integer) session.save(group);
Group groupGot = (Group) session.get(Group.class, groupId);
```

JDO 2.0

- Especificación JSR 243 → Agosto 2005
- Algunas mejoras
  - Mejora de JDOQL: grouping, having by, subset, uniqueness, Código específico de proveedor, SQL Nativo
  - Attach / Detach: permite usar objetos persistentes fuera de la transacción, e identificar si un objeto ha de ser creado o actualizado
- Implementación de referencia → JPox
- Sigue teniendo el mismo problema → Poco peso político.
  - Especificación aprobada con el voto en blanco de JBoss (Hibernate)

EJB 3

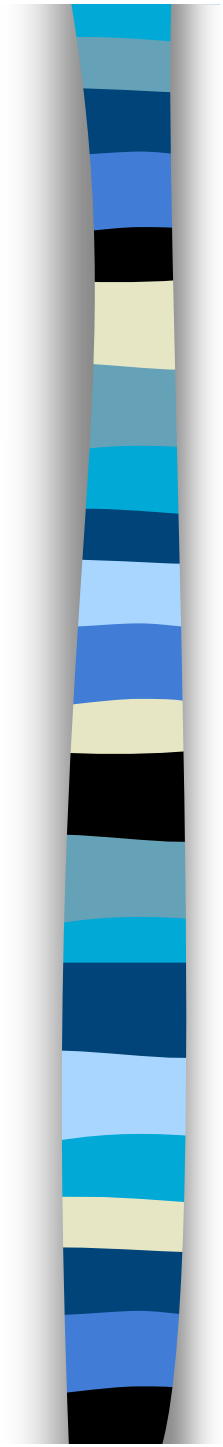
- Especificación JSR 220 → Diciembre 2005
- Sun decide unir los esfuerzos de los componentes de EJB3 (parte de persistencia) y JDO2 para crear una **única** API con las ventajas de las anteriores
  - Hibernate, JDO, TopLink
- La especificación de la persistencia se hace independiente (en un documento aparte) → **JPA**
- Colaboradores de todas las empresas importantes
  - BEA, Oracle, JBoss, Borland, IBM, Sun.. Google!
- Ya existen implementaciones (e.g. Hibernate3)

JPA (EJB3) - Objetivos

- **ESTANDAR**
- Una única API para la Comunidad Java, que recoja lo mejor de las anteriores
- **Reducir al máximo la complejidad**
- Modelo de persistencia no intrusivo al estilo **POJO**, que funcione **tanto en J2EE como en J2SE**
- Permitir a los usuarios de JDO2 una fácil migración a EJB3

JPA (EJB3) - Novedades

- **No precisa de contenedor**! Funciona tanto en J2EE como J2SE
- Puede usarse independientemente del resto de los servicios (transacciones, seguridad, ..)
- Metadata con **Anotaciones**: Se eliminan los deployment descriptors, se reduce drásticamente el número de clases a crear (o generar)
- Configuraciones por defecto: reducen la cantidad de configuración a especificar
- No Intrusión: los objetos a persistir (Entity Beans) no necesitan implementar interfaces EJB
- Herencia y poliformismo
- Lenguaje de consulta (EJBQL) mejorado: inner and outer join, operaciones bulk, sql nativo..



JPA (EJB3)

- Ventajas
  - **Testing**
  - Simplicidad: una única clase para declarar la persistencia (con la ayuda de anotaciones)
  - Facilidad de aprendizaje
  - Transparencia: las clases a persistir son simples POJOs
  - No hay restricciones con respecto a relaciones entre objetos (herencia, poliformismo)
- Desventajas
  - Anotaciones
    - Descripción de la Base de Datos en el código
    - Solución: Siempre se puede utilizar un XML

JPA (EJB3) – Ejemplo Conf.

- META-INF/persistence.xml

```
<persistence>
  <persistence-unit name=„ua_manager" transaction-
type="RESOURCE_LOCAL">
    <class>es.ua.ejb3.User</class>
    <class>.....</class>
    <properties>
      <property name="hibernate.dialect"
          value="org.hibernate.dialect.HSQLDialect" />
      <property name="hibernate.connection.driver_class"
          value="com.mysql.jdbc.Driver" />
      <property .... />
      <!-- cache -->
    </properties>
  </persistence-unit>
</persistence>
```

JPA (EJB3) – Ejemplo

```java
@Entity
@Table (name="ua_user")
public class User {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;
    @Column(name="name")
    private String name;
    @Column(name="email")
    private String email;
    @Column(name="created_on")
    private Date createdOn;

    //getters and setters
}
```

```java
@Entity
@Table (name="ua_group")
public class Group {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;
    @Column(name="name")
    private String name;
    @OneToMany(fetch = FetchType.EAGER)
    @JoinTable(
            name="ua_user_group",
            joinColumns = { @JoinColumn( name="group_id") },
            inverseJoinColumns = {@JoinColumn( name="user_id")}
    )
    private Set<User> users;

    //getters and setters
}
```

JPA (EJB3) – Ejemplo

- Usamos un EntityManager para manipular los objetos persistentes

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("ua_manager");
EntityManager em = emf.createEntityManager();
```

- Transacciones

```
em.getTransaction().begin();
//codigo..
em.getTransaction().commit();
```

- Cerrar el EntityManager

```
em.close();
```

JPA (EJB3) – Ejemplo

- Persistir

```
User user = new User("ruben", "hola");
//el identificador se asocia al objeto automáticamente
em.persist(user);
```

- Obteniendo objetos

```
User user1 = em.find(User.class, 1);
```

```
//obtener una referencia cargará el objeto cuando accedamos a sus propiedades
User user2Reference = em.getReference(User.class, 2);
```

```
Query newUsersQuery = em.createQuery("from User where createdOn > ?1");
newUsersQuery.setParameter(1, date);
List<User> newUsers = newUsersQuery.getResultList();
```

EJBQL mejorado:
proyeccion, resultados escalares, named parameters, paginación, named queries, native queries, query hints (max size, timeout, cache..)

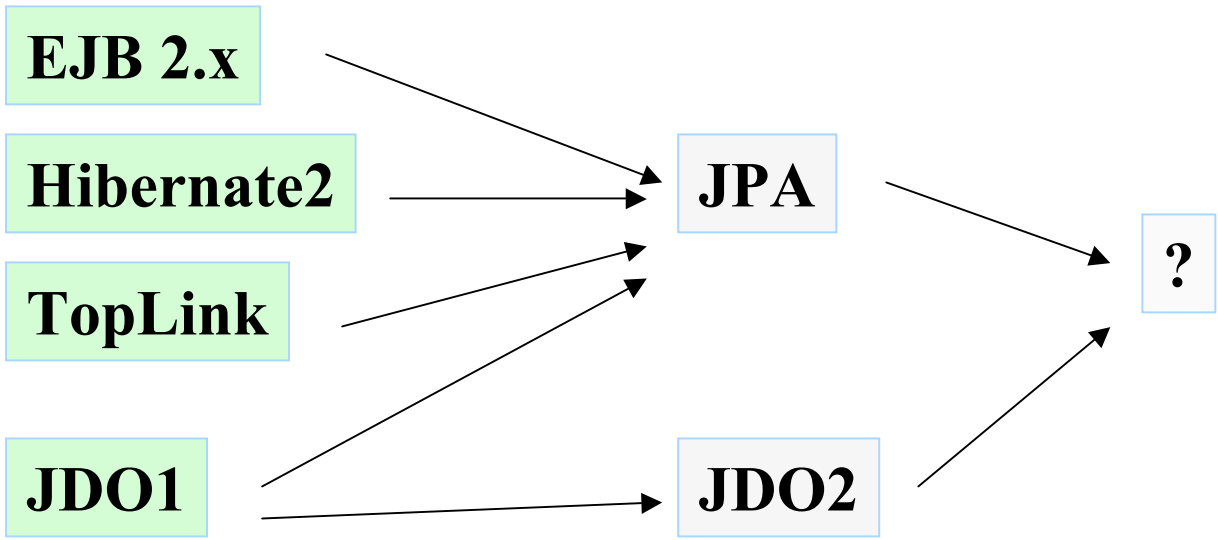
- Update

```
user.setEmail("otro@otro.com");
em.flush();
```

Delete

```
em.remove(user1);
```

Futuro ¿?

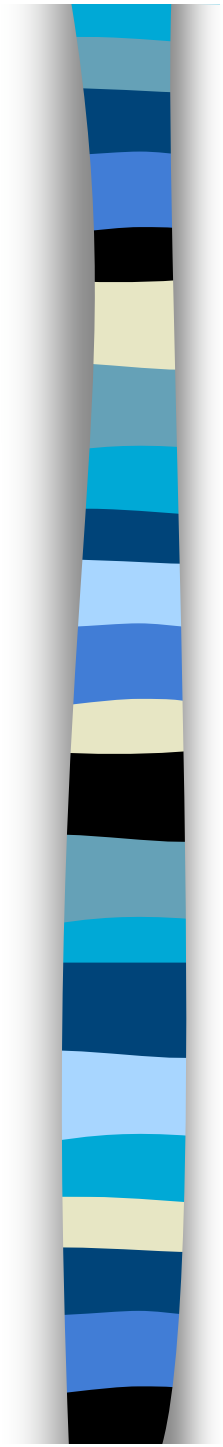


¿Cuál es el mejor?

- Necesidades del proyecto
- Tamaño del proyecto
- Necesidades de la organización
- Experiencia de los desarrolladores
- Madurez de la tecnología
- Documentación y Soporte

Características a tener en cuenta

- Curva de aprendizaje
- Facilidad de Debug (errores SQL)
- Rendimiento
- Facilidad de Uso
- Cantidad de Trabajo
- Testeabilidad
- Integración
- Intrusión
- Reusabilidad
- Soporte para Transacciones
- Escalabilidad
- Facilidad de Refactorización
- Seguridad
- Persistencia transitiva (esilo de cascada)
- Herramientas de apoyo (e.g. generación de esquema BD)



Consejos

- Investigar y comparar
- Lo último no es siempre lo mejor (e.g. EJB)
- Cada proyecto es diferente
- Rendimiento vs Facilidad de desarrollo
- SQL sigue siendo necesario
- Siempre podemos mezclar
  - Considerar usar un framework de aplicaciones y abstraer acceso (DAO pattern)
- Considerar Alternativas
  - ¿Necesitamos una base de datos relacional?
  - XML, XMLBD
  - Bases de datos orientadas a objetos


Gracias...