**Curso de Programación JAVA EE**

**Instructor: Goicoechea, Lucas**

**contenidos**

**Acceso directo a Base de Datos (JDBC)**

**Patrón de Diseño DAO**

**Impedance Mismatch**

**Mecanismos de Persistencia**

**API de Persistencia (JPA)**

**EntityBean - POJO - EntityManager**

**ORM**

**Hibernate**

Contenido

Mecanismos de Persistencia

* Acceso directo a Base de datos (JDBC)
* Problema: Impedance Mismatch
* Mapeador Objeto Relacional (JPA)

JDBC

Acceso directo a Base de Datos

¨ Ventaja

Buen desempeño debido a que el acceso a la base se realiza en forma directa

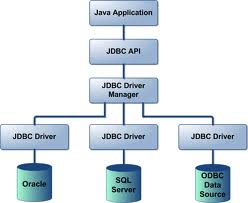
¨ Desventaja

Mayor complejidad para el diseño de la persistencia.

¨ Herramienta Java

JDBC (Java Database Connectivity)

**Arquitectura JDBC**

****

JDBC - Diseño

Patrón de Diseño DAO:

¨ Encapsula todo el acceso a una fuente de datos utilizando la capa de acceso a datos.

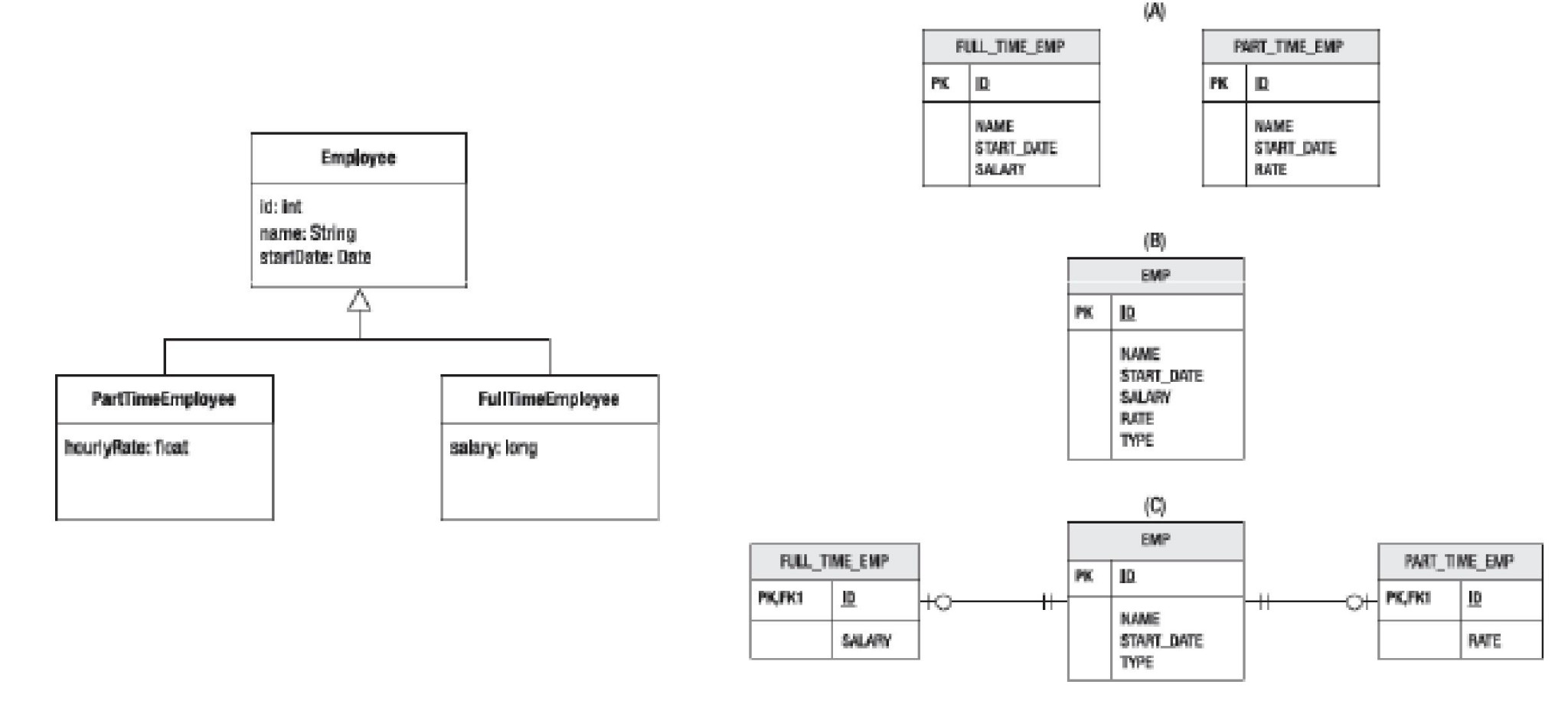
¨ Cada objeto de negocio a ser persistido crea un DAO con la información necesaria.

¨ Luego utiliza el objeto DAO para obtener y guardar información del origen de datos.

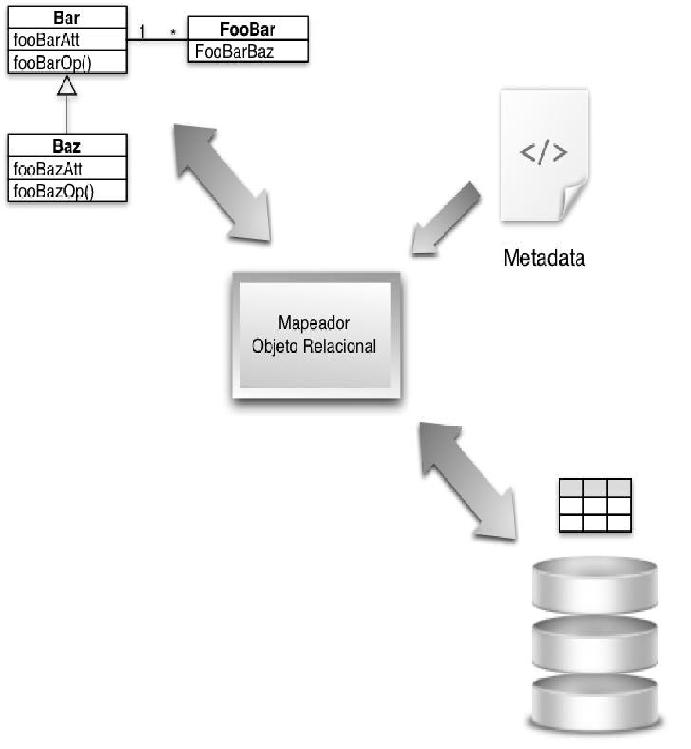
Problemas al tener que “mapear” entidades -> Impedance Mismatch

La diferencia entre el modelo relacional y el modelo orientado a objetos, se denomina Impedance Mismatch, surge debido a las similaridades, sino a las diferencias entre ambos modelos.

**Un ejemplo:**

****

Mapeador Objeto - Relacional



Mapeador Objeto - Relacional

¨ Ventaja

Reduce la cantidad de código necesario para lograr la persistencia. La aplicación resultante es fácil de mantener.

¨ Desventaja

Requiere Mantener metadata sobre las entidades.

¨ Herramienta Java - conocidos como ORM

Hibernate

OpenJPA

API de Persistencia

de Java (JPA)

API estandar utilizado para la mayoría de los ORM o re-implementado por otros. La idea es que el código Java tenga la definición o la referencia de esta API (si se quiere luego se podría cambiar o re-implementar).

JPA – Conceptos - EntityBean

* Se anotan con @Entity
* Tienen una propiedad anotada con @Id
* Constructor sin argumentos public/protected
* No puede ser final
* Puede extender de otra
* Puede ser abstracta
* Es un POJO (Plain Old Java Objects), objeto liviano que no implementan ninguna interfaz

@Entity

public class Team {

private int id;

private String name;

private Collection players;

@Id

public int getId() {

return id;

}

public void setId( int value ) {

id = value;

}

public String getName() {

return name;

}

public void setName( String value ) {

name = value;

}

@OneToMany

public Collection getPlayers() {

return players;

}

public void setPlayers( Collection value ) {

players = value;

}

}

Relaciones entre entidades

Relaciones Unidireccionales

¨ Sólo se puede navegar desde una entidad a la otra

Relaciones Bidireccionales

¨ Desde cualquiera de las dos entidades es posible navegar a la otra

Tipos

¨ one-to-one

¨ one-to-many

¨ many-to-one

¨ many-to-many

**MAPEANDO RELACIONES - BÁSICO**

@Entity

public class **Customer** implements Serializable{

@Id

@Column(name="CUSTOMER\_ID")

private int id;

@Column(name="FIRST\_NAME")

private String firstName;

@Column(name="LAST\_NAME")

private String lastName;

@OneToOne //UNO A UNO

private Referee referee;

//UNO A MUCHOS

// DONDE FETCH FUERZA A LEER CUANDO SE INVOCA

@OneToMany(mappedBy="customer", fetch=FetchType.EAGER)

private List accounts;

//MUCHOS A MUCHOS

@ManyToMany

// NOTA: Esta annotation no es necesaria, pero si sirve para

// dejar mas prolija la entidad que relaciona a las tablas customer y

// address

@JoinTable(name="CUSTOMER\_ADDRESS",

joinColumns=@JoinColumn(name="CUST\_ID",

referencedColumnName="CUSTOMER\_ID"

), inverseJoinColumns=@JoinColumn(name="ADD\_ID",

referencedColumnName="ADDRESS\_ID")

)

private List addresses;

@Column(name="GENDER", length=20)

@Enumerated(EnumType.STRING)

private Gender gender;

El mapeo se puede hacer de la misma forma que con annotations en el archivo orm.xml.  
Adjunto el link al XSD del orm.xml:  
<http://java.sun.com/xml/ns/persistence/orm_1_0.xsd>  
No es necesario poner el mapeo en el persistence.xml y tampoco caer en un hbm.xml de Hibernate.

Entity Manager / SessionManager (dirty)

El API del EntityManager es la parte más importante e interesante de JPA.

Maneja el ciclo de vida de las entidades.

Actúa de puente entre el mundo O.O y el mundo relacional.

-Metodos-

public void persist(Object entity);

public <T> T merge(T entity);

public void remove(Object entity);

public <T> T find(Class<T> entityClass,

Object primaryKey);

public void flush();

public void setFlushMode(FlushModeType flushMode);

public FlushModeType getFlushMode();

public void refresh(Object entity);

Tambien posee un lenguaje de OO a SQL -> JPQL

public Query createQuery(String jpqlString);

public Query createNamedQuery(String name);

Donde jpqlString podría ser: “Select \* from Empleado e where e.nombre=:parametroNombre”

JPA es un API, una serie de interfaces, y por los tanto tenemos que trabajar con un proveedor de persistencia que implemente dicha interfaz. Actualmente, en el mercado existen varios proveedores de persistencia JPA como Hibernate, iBatis, TopLink, etc…

**Hibernate**

Trabajar con software orientado a objetos y bases de datos relacionales puede hacernos invertir mucho tiempo en los entornos actuales. Hibernate es una herramienta que realiza el *mapping* entre el mundo orientado a objetos de las aplicaciones y el mundo entidad-relación de las bases de datos en entornos Java. El término utilizado es ORM (object/relational mapping) y consiste en la técnica de realizar la transición de una representación de los datos de un modelo relacional a un modelo orientado a objetos y viceversa.

Hibernate no solo realiza esta esta transformación sino que nos proporciona capacidades para la obtención y almacenamiento de datos de la base de datos que nos reducen el tiempo de desarrollo.

Ejemplo Mapping

package prueba;

public class Persona {

private Long idPersona;

private String nombre;

private Date fechaNacimiento;

public Date getFechaNacimiento() {

return fechaNacimiento;

}

public void setFechaNacimiento(Date fechaNacimiento) {

this.fechaNacimiento = fechaNacimiento;

}

public Long getIdPersona() {

return idPersona;

}

public void setIdPersona(Long idPersona) {

this.idPersona = idPersona;

}

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

}

Persona.hbm.xml (Archivo Mapping)

<hibernate-mapping package=“prueba">

<class name=“Persona" table=“PERSONA">

<id name=“IdPersona" column="ID\_PERSONA">

<generator class="native"/>

</id>

<property name="nombre“ not-null=“true”/>

<property name=“fechaNacimiento"/>

</class>

</hibernate-mapping>

Links prácticos:

http://pajarokillo.wordpress.com/2011/11/03/jpa-2-0-introduccion/

http://pajarokillo.wordpress.com/2011/11/17/jpa-2-0-mapeando-relaciones-one-to-one/

http://www.davidmarco.es/blog/entrada.php?id=144

http://java-all-frameworks.blogspot.com.ar/2011/06/jpa-utilidades-y-ejemplos-para.html

http://es.scribd.com/doc/46308101/Introduccion-a-JPA

http://schuchert.wikispaces.com/JPA+Tutorial+1+-+Getting+Started

http://es.scribd.com/doc/71092204/6/Impedance-Mismatch