# OBJECT RELATIONAL MAPPING (ORM)

Persistencia Orientada a Objetos en BDs Relacionales

# HASTA HACE UN RATITO ERAN FELICES

Era todo muy simple y transparente! **¿Por qué?**

*Porque persistíamos en una estructura física idéntica a la estructura que manejamos en memoria.*

Si utilizamos una BD relacional la estructura física cambia y tenemos que trabajar un poco más.

# DIFERENCIA DE IMPEDANCIA

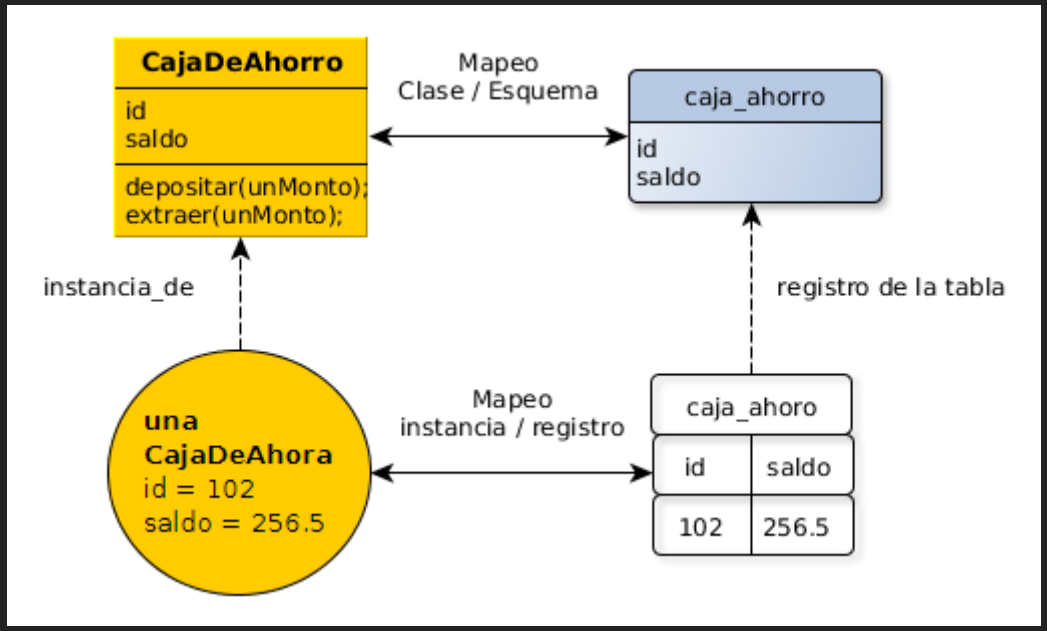
Se da cuando desarrollamos en un modelo (ej: objetos) y persistimos en otro modelo (ej: tablas/relaciones).

- En Objetos navegamos a través de colaboradores vs los joins en relacional.
- En Objetos tenemos colecciones con su semántica (ej: Set), en relacional las relaciones y las restricciones.
- En Objetos tenemos Herencia y polimorfismo, en el relacional relaciones.

# DIFERENCIA DE IMPEDANCIA

*En base a estas diferencias, el ORM nos ayuda mapeando el mundo de objetos al relacional, de modo de poder disfrutar de la persistencia transparente.*

# ¿QUÉ MAPEAMOS?




- Clase con Tabla
- Instancia con Tupla
- Atributo con Columna

# JAVA PERSISTENCE API

Estándar creado en 2006 por la Java Community Process (JCP)

¿JDO y JPA?

Veamos su API, o sea las interfaces y clases ubicadas en: *javax.persistence*

# JAVA PERSISTENCE API (JPA)

## ESTRUCTURA DE UNA TRANSACCIÓN

```
EntityManagerFactory emf =
          Persistence.createEntityManagerFactory("jpa-derby");
EntityManager em = emf.createEntityManager();
EntityTransaction tx = em.getTransaction();
try {
 tx.begin();
 //hacer algo con em
 tx.commit();
} catch (Exception e) {
 tx.rollback();
 throw new RuntimeException(e);
} finally {
 if (em != null && em.isOpen())
  em.close();
 if (emf != null)
```

¿Cuál es el Contexto de Persistencia?

# JAVA PERSISTENCE API (JPA)

## ETC/META-INF/PERSISTENCE.XML

```
<!--?xml version="1.0"?-->
<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:s
 <persistence-unit name="jpa-derby">
  <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
   <properties>
    <property name="javax.persistence.jdbc.driver" value="org.apache.derby.jdbc.ClientDriver">
    <property name="javax.persistence.jdbc.url" value="jdbc:derby://localhost:1527/bd-name">
    <property name="javax.persistence.jdbc.user" value="user">
    <property name="javax.persistence.jdbc.password" value="pass">
    <property name="hibernate.dialect" value="org.hibernate.dialect.DerbyDialect">
    <property name="hibernate.hbm2ddl.auto" value="create">
    <property name="hibernate.show_sql" value="true">
   </property></property></property></property></property></property></property></properties>
 </persistence-unit>
</persistence>
```

¿hibernate.hbm2ddl.auto = create?

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: ENTIDADES

Los objetos persistentes deben **anotarse** con *javax.persistence.Entity*

```
@Entity
public class Persona {
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 private Long id;
 private String nombre;
 ...
 //no-arg constructor requerido
 protected Persona() { }
 ...
 //get-set privados
}
```

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: MAPEOS

### Muchos a Uno (Una *Persona* una *Direccion*)

```
@Entity
public class Persona {
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 private Long id;
 private String nombre;
 @ManyToOne
 private Address direccion;
 ...
}
```

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: MAPEOS

### Muchos a Uno (Una *Persona* una *Direccion*)

```
@Entity
public class Address {
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 private Long id;
 private String direccion;
 ...
}
```

# JAVA PERSISTENCE API (JPA)

## MUCHOS A UNO (UNA *PERSONA* UNA *DIRECCION*)

Este mapeo genera las siguientes tablas:

```
persona(id, nombre, direccion_id);

direccion(id, direccion);
```

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: MAPEOS

### Uno a Muchos (Una *Persona* muchos *Telefonos*)

```
@Entity
public class Persona {
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 private Long id;
 private String nombre;

 @OneToMany
 @JoinColumn(name = "id_persona")
 private Collection telefonos = new ArrayList<>();
 ...
}
```

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: MAPEOS

### Uno a Muchos (Una *Persona* muchos *Telefonos*)

```
@Entity
public class Telefono {
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 private Long id;
 private String numero;
 ...
}
```

# JAVA PERSISTENCE API (JPA)

## UNO A MUCHOS (UNA *PERSONA* MUCHOS *TELEFONOS*)

Este mapeo genera las siguientes tablas:

```
persona(id, nombre);

telefono(id, numero, id_persona);
```

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: MAPEOS

### Componentes (Una *Persona* una *Direccion*)

```
@Entity
public class Person {
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 private Long id;
 private String nombre;
 @Embedded
 private Address address;
```

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: MAPEOS

Componentes (Una *Persona* una *Direccion*)

```
@Embeddable
public class Address {
 private String address;
 ...
}
```

Note que no es una Entidad... y por lo tanto no requiere ID

# JAVA PERSISTENCE API (JPA)

## COMPONENTES (UNA *PERSONA* UNA *DIRECCION*)

Este mapeo genera la siguiente tabla:

```
persona(id, nombre, direccion);
```

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: MAPEOS

Bidireccional (Una *Persona* varios *Telefonos*, un *Telefono* una *Persona*)

```
@Entity
public class Telefono {
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 private Long id;

 @ManyToOne
 @JoinColumn(name = "id_persona")
 private Persona persona;
 ...
}
```

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: MAPEOS

Bidireccional (Una *Persona* varios *Telefonos*, un *Telefono* una *Persona*)

```
@Entity
public class Persona {
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;

@OneToMany(mappedBy = "persona")
private Collection telefonos = new ArrayList<>();
...
}
```

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: MAPEOS

Bidireccional (Una *Persona* varios *Telefonos*, un *Telefono* una *Persona*)

El *mappedBy* define que la navegabilidad será utilizando la columna definida en *Telefono.persona*

Le dice al ORM: No generes otra columna... usa la que se mapeó en *Telefono.persona*

Si no me creen... prueben Ustedes que sucede si no ponen *mappedBy*

# JAVA PERSISTENCE API (JPA)

Bidireccional (Una *Persona* varios *Telefonos*, un *Telefono* una *Persona*)

Este mapeo genera las siguientes tablas:

```
persona(id, nombre);

telefono(id, numero, id_persona);
```

# JAVA PERSISTENCE API

¿Y la Herencia? ¿Cómo se mapea?

# JAVA PERSISTENCE API (JPA)

## HERENCIA

Supongamos que queremos persistir la siguiente jerarquía de herencia:

```
public abstract class CuentaBancaria {
 protected float monto;
 public abstract float extraer(float monto);
 public void depositar(float anAmount) {
         this.monto += anAmount;
 }
}
```

# JAVA PERSISTENCE API (JPA)

## HERENCIA

```
public class CajaDeAhorro extends CuentaBancaria {
 ...
 public float extraer(float monto) {
  @Override
  public float extraer(float unMonto) {
   if (this.monto >= unMonto) {
    this.monto -= unMonto;
    return this.monto;
   }
   throw new RuntimeException("No hay suficiente dinero para... ");
  }
 }
 ...
}
```

# JAVA PERSISTENCE API (JPA)

## HERENCIA

```
public class CuentaCorriente extends CuentaBancaria {
 private float descubierto;
 ...
 @Override
 public float extraer(float unMonto) {
  if (unMonto <= (this.monto + descubierto)) {
   this.monto -= unMonto;
   return this.monto;
  }
  throw new RuntimeException("No hay suficiente dinero para... ");
 }
 ...
}
```

# JAVA PERSISTENCE API (JPA)

## HERENCIA: TRES FORMAS DE MAPEARLA

- Single Table (default)
- Joined
- Table per Concrete Class

# JAVA PERSISTENCE API (JPA)

## SINGLE TABLE

```
@Entity
@Inheritance
@DiscriminatorColumn(name = "tipo_cuenta")
public abstract class CuentaBancaria {
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 private Long id;
 ...
}
@Entity
@DiscriminatorValue("CC")
public class CuentaCorriente extends CuentaBancaria { ... }
@Entity
@DiscriminatorValue("CA")
public class CajaDeAhorro extends CuentaBancaria { ... }
```

# JAVA PERSISTENCE API (JPA)

## SINGLE TABLE

Genera una única tabla con una columna que indica de que clase es la instancia persistida:

```
cuentabancaria(tipo_cuenta, id, monto, descubierto);
```

Si persistimos estas instancias:

```
new CajaDeAhorro(10.000);new CuentaCorriente(5000, 600);
```

Nos quedan las tuplas:

```
cuentabancaria("CA", 1L, 10.000, null);
cuentabancaria("CC", 2L, 5000, 600);
```

# JAVA PERSISTENCE API

y... ¿cómo persistimos y recuperamos objetos de la BD?

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: GUARDAR

```
EntityManager em = emf.createEntityManager();
EntityTransaction tx = em.getTransaction();
try {
 tx.begin();
 Persona p = new Persona("Ignacio");
 em.persist(p);
 tx.commit();
} catch (Exception e) {
 tx.rollback();
 throw new RuntimeException(e);
} finally {
 if (em != null && em.isOpen())
  em.close();
}
```

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: GUARDAR

```
EntityManager em = emf.createEntityManager();
EntityTransaction tx = em.getTransaction();
try {
 tx.begin();
 Persona p = new Persona("Ignacio", new Address("San Martin 346"));
 em.persist(p);
 tx.commit();
} catch (Exception e) {
 tx.rollback();
 throw new RuntimeException(e);
} finally {
 if (em != null && em.isOpen())
  em.close();
}
```

¿Persiste la instancia de *Address* también? Solo si la mapeo con cascade.

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: CASCADE

```
@Entity
public class Persona {
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 private Long id;
 @ManyToOne(cascade = CascadeType.PERSIST)
 private Address direccion;
 ...
}
```

De esta forma, cuando persisto una *persona*, persiste su *direccion* también

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: RECUPERAR

*find*() vs *getReference*()

```
...
try {
 tx.begin();
 Persona p = em.getReference(Persona.class, 1L);
 Persona p = em.find(Persona.class, 1L);
 System.out.println(p.nombre());
 tx.commit();
...
}
```

La diferencia esta en cuándo se ejecuta la query para traer la *persona* **1L** de la BD a memoria.

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: RECUPERAR / GUARDAR

*getReference*(): ¿Para que se usa?

```
...
try {
 tx.begin();
 Persona p = em.getReference(Persona.class, 2L);
 p.direccion(new Address("San Martin 356"))
 tx.commit();
...
}
```

Si lo hiciera con *find*(), además del *insert* de *address* y el *update* en *persona*, se haría un *select*.

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: EARLY VS LAZY

Las relaciones ManyToOne son Early por defecto

```
...
try {
 tx.begin();
 Persona p = em.find(Persona.class, 2L);
 ...
 tx.commit();
...
}
```

Entonces, ¿si traigo la *persona* **2L**, trae su *dirección*?
Sabiendo que estan mapeados con ManyToOne.

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: EARLY VS LAZY

Si quiero hacer **Lazy** una relación ManyToOne, debo especificarlo:

```
@Entity
public class Persona {
 @Id
 @GeneratedValue(strategy = GenerationType.AUTO)
 private Long id;
 @ManyToOne(fetch = FetchType.LAZY)
 private Address direccion;
 ...
}
```

# JAVA PERSISTENCE API (JPA)

## OBJETOS PERSISTENTES: EARLY VS LAZY

Las relaciones OneToMany son Lazy por defecto

O sea que si traigo una *persona*, no traigo sus *telefonos* a menos que los pida.

Existe además **extra-lazy**, ¿qué es?