Universidad de los Andes

Persistencia
Java Persistence API – JPA 2.0

Por: Rafael Gustavo Meneses M.Sc.

Departamento de Ingeniería de Sistemas y Computación
Especialización en Construcción de Software
Bogotá, COLOMBIA

Universidad de los Andes

Agenda

- Introducción
- Generalidades JPA
- Entities (Entidades)
  - Relaciones entre Entidades
  - Mapeo a la Base de Datos
  - Administración de Entidades
- Unidades de Persistencia
- Referencias

Universidad de los Andes

Agenda

- **Introducción**
- Generalidades JPA
- Entities (Entidades)
  - Relaciones entre Entidades
  - Mapeo a la Base de Datos
  - Administración de Entidades
- Unidades de Persistencia
- Referencias

Object/Relational Mapping (ORM)

- Una parte importante de cualquier proyecto de desarrollo de aplicaciones empresariales es la capa de persistencia
  - Acceso y manipulación de datos persistentes → Base de datos relacional (RDBMS)
- ORM convierte datos entre sistemas incompatibles:
  - **RDB** → Datos organizados en bases de datos relacionales como filas y columnas en tablas
  - **Objetos** → Lenguajes OO como Java

Queremos trabajar con objetos,
no con filas y columnas de tablas

Universidad de los Andes

Agenda

- Introducción
- **Generalidades JPA**
- Entities (Entidades)
  - Relaciones entre Entidades
  - Mapeo a la Base de Datos
  - Administración de Entidades
- Unidades de Persistencia
- Referencias

Características de JPA

- Framework de mapeo O/R estándar de Java EE
- Framework ORM que permite persistencia transparente a través de POJOs (**P**lain **O**ld **J**ava **O**bjects):
  - Permite trabajar sin estar restringido por el modelo de base de datos relacional basado en tablas → Maneja la incompatibilidad objeto/relación entre ambos sistemas
- Permite construir objetos persistentes mediante conceptos comunes de POO

¿Qué es JPA?

- "The Java Persistence API provides a POJO persistence model for object-relational mapping. [JPA] was developed by the EJB 3.0 software expert group as part of JSR 220, but its use is not limited to EJB software components. It can also be used directly by web applications and application clients, and even outside the Java EE platform, for example, in Java SE applications." [1]

- JPA facilita el uso de POJOs como entidades persistentes reduciendo significativamente la necesidad de *descriptores complejos* y *helpers extras* → *No se necesitan interfaces*

¿Qué es JPA?

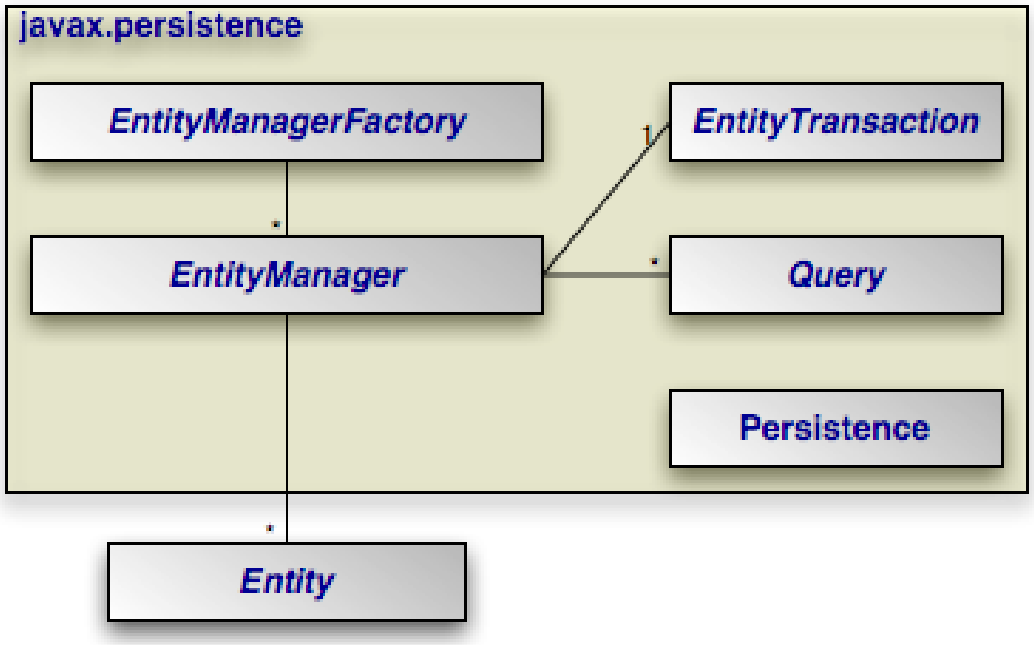
- JPA mapea objetos a la base de datos por medio de *metadata*
- Cada entidad tiene asociado la metadata que describe su respectivo mapeo O/R

- Metadata puede ser escrita de dos (2) formas:
  - **Anotaciones →** El código es anotado directamente con las anotaciones descritas en el paquete **`javax.persistence`**
  - **Descriptores XML →** El mapeo se define en archivos XML que son deplegados junto con las entidades

Ventajas de JPA

- Simplifica el modelo de persistencia → Modelo liviano basado en POJOs
- El mapeo O/R está completamente basado en *metadata*
- Elimina el uso de los complejos DAOs (Data Access Objects)
- El API ayuda al manejo de transacciones
- Soporta modelos de dominio ricos en términos de objetos: herencia, polimorfismo
- Unifica el modelo de persistencia de Java → Puede ser usado en aplicaciones de escritorio y no solamente en aplicaciones empresariales
- Proporciona un lenguaje de consulta común (JPQL) que evita escribir consultas directamente en SQL

Principales Componentes de JPA



Tomado de [2]

Principales Componentes de JPA

- **Persistence →** Contiene métodos de clase (estáticos) para obtener instancias de *EntityManagerFactory* independientemente del proveedor
- **EntityManagerFactory →** Fábrica para crear *EntityManagers*
- **EntityManager →** Principal interface JPA utilizada por las aplicaciones
  - Usada para manejar un conjunto de objetos persistentes
  - Tiene APIs para insertar nuevos objetos y borrar existentes
  - Actúa como fábrica para instancias de *Queries*
- **Entity →** Objetos persistentes
- **EntityTransaction →** Manejo de transacciones entre entities
- **Query →** Interface implementada por cada proveedor para encontrar objetos persistentes

Universidad de los Andes

Agenda

- Introducción
- Generalidades JPA
- **Entities (Entidades)**
  - Relaciones entre Entidades
  - Mapeo a la Base de Datos
  - Administración de Entidades
- Unidades de Persistencia
- Referencias

¿Qué son las Entities?

- No son **EntityBeans** !!!
  - No son thread-safe → No es un problema dentro de un contenedor JavaEE
- Son POJOs + Anotaciones
  - No pueden invocarse remotamente, los métodos se ejecutan localmente
- Una entidad representa una tabla en una base de datos relacional, cada instancia corresponde a un fila en dicha tabla
- El principal artefacto de programación de una entidad es la *clase de entidad*
  - Las entidades pueden utilizar *helpers*

Ejemplo

```
@Entity
public class Customer implements Serializable {
   private Long id;
   private String name;
   private Address address;

   // No argument constructor
   public Customer() {}

    @Id
    public Long getID() {
       return id;
    }

    private void setID (Long id) {
         this.id = id;
    } ...
```

Clase de Entidad

- Debe tener la anotación *javax.persistence.Entity* y en ciertos casos implementar *Serializable*

```
@Entity(access = AccessType.FIELD)
public class miEntidadBean implements Serializable
```

- El estado persistente de una entidad está representado por:
  - **Field Access** → Se especifica en los atributos. Todos los atributos no transcientes persisten
  - **Property Access** → Se especifica en los métodos *getter.* Todas las propiedades públicas, protegidas y no transcientes persisten

Clase de Entidad

- Pueden extender clases de entidad o no entidad, una clase que no es de entidad puede extender una clase de entidad
- Debe tener un constructor público sin parámetros, puede tener más constructores
- No debe tener campos, atributos ni métodos declarados *final*
- Todo atributo persistente debe ofrecer métodos get/set
- Los atributos de tipo colección deben usar las interfaces del framework java.util.Collection con generics de Java 5 (Collection, Set, Map, etc.)

Identificación de Instancias

- Cada instancia de una entidad tiene una *llave primaria* que lo identifica
- Las llaves primarias simples usan la anotación *javax.persistence.Id* en el método *get* (o en el atributo) para señalar que un atributo es la llave primaria
- Tipos válidos → *Primitivos*, *Wrappers*, *String*, *java.util.Date*, *java.sql.Date*, *java.math.BigDecimal*, *java.math.BigInteger*

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
public Long getId() {
   return id;
}
```

Identificación de Instancias (2)

- Es posible definir clases compuestas para llaves primarias → Deben implementar *Serializable*

```
@Entity(access = AccessType.FIELD)
@IdClass(com.example.PersonPK.class)
public class Person {
   @Id
   String firstName;

   @Id
   String lastName;
   ...
}
```

Universidad de los Andes

Agenda

- Introducción
- Generalidades JPA
- **Entities (Entidades)**
  - **Relaciones entre Entidades**
  - Mapeo a la Base de Datos
  - Administración de Entidades
- Unidades de Persistencia
- Referencias

Universidad de los Andes

Relaciones entre Entities

- **One-to-one**
  - Cada instancia de una entidad está relacionada a una única instancia de otra
- **One-to-many**
  - Una instancia de una entidad puede estar relacionada a múltiples instancias de otra
- **Many-to-one**
  - Múltiple instancias de una entidad pueden estar relacionadas a una única instancia de otra
- **Many-to-many**
  - Las instancias de una entidad pueden estar relacionadas a múltiples instancias de otra

Funcionamiento de las relaciones

- La dirección de una relación puede ser bidireccional o unidireccional
- Una relación tiene un lado dueño (owner) → Determina cómo se podrán propagar las modificaciones
- Las relaciones bidireccionales tienen además un lado inverso
- Una relación es marcada en un atributo usando las anotaciones
  - *javax.persistence.OneToOne*
  - *javax.persistence.OneToMany*
  - *javax.persistence.ManyToOne*
  - *javax.persistence.ManyToMany*

Funcionamiento de las relaciones (2)

- Carga en memoria de las relaciones → Indica el momento en el cual los datos asociados al campo relacionado con otra entidad son cargados en memoria
  - **EAGER** → Cuando se carga la clase dueña que tiene la relación
  - **LAZY** → Cuando se utiliza el campo que representa la relación
- Ejemplos:

```
@ManyToMany(fetch = FetchType.EAGER)
public Collection<Producto> getProductos()

@OneToOne
public Usuario getUsuario()
```

Relaciones Bidireccionales

- Se puede navegar en ambas direcciones
- Deben cumplir las siguientes reglas:
  - El lado inverso de la relación debe referenciar el lado dueño usando el elemento *mappedBy* en la anotación de la relación
  - En la relación *Many-to-one* o Many-to-Many el lado *Many* es el dueño
- En las relaciones One-to-One el dueño corresponde al lado que tiene la llave foránea

Relaciones Bidireccionales (2)

- Ejemplo:

Customer.java

```
@OneToMany (mappedBy = "customer")
public Collection<Order> getOrders()
```

Order.java

```
private Customer customer;
@ManyToOne
public Customer getCustomer()
```

Relaciones Bidireccionales (3)

- Las entidades relacionadas pueden tener dependencias de existencia
- Las operaciones a propagar corresponden a *DETACH*, *MERGE*, *PERSIST*, *REFRESH*, *REMOVE* y *ALL*
- Las relaciones candidatas a especificar la propagación son **`@OneToOne`** y **`@OneToMany`**
- Las entidades inversas especifican en la relación el valor de propagación utilizando el elemento *cascade=javax.persistence.CascadeType.ALL*

Customer.java

```
@OneToMany(cascade=CascadeType.ALL, mappedBy="customer")
public Collection<LineItem> getItems(){
    return items;
}
```

Relaciones Bidireccionales (4)

@OneToOne

```
@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface OneToOne {
    Class targetEntity() default void.class;
    CascadeType[] cascade() default {};
    FetchType fetch() default EAGER;
    boolean optional() default true;
    String mappedBy() default "";
}
```

Relaciones entre Entities

```
@Entity
public class User {
    @Id
    protected String userId;
    protected String email;
    @OneToOne
    protected BillingInfo billingInfo;
}

@Entity
public class BillingInfo {
     @Id
     protected Long billingId;
     protected String creditCardType;
     protected String creditCardNumber;
     protected String nameOnCreditCard;
     @OneToOne(mappedBy="billingInfo", optional="false");
     protected User user;
}
```

Tomado de [4]

Relaciones Bidireccionales (6)

@OneToMany / @ManyToOne

```
@Target({METHOD, FIELD}) @Retention(RUNTIME)
public @interface OneToMany {
    Class targetEntity() default void.class;
    CascadeType[] cascade() default {};
    FetchType fetch() default LAZY;
    String mappedBy() default "";
}

@Target({METHOD, FIELD}) @Retention(RUNTIME)
public @interface ManyToOne {
    Class targetEntity() default void.class;
    CascadeType[] cascade() default {};
    FetchType fetch() default EAGER;
    boolean optional() default true;
}
```

Universidad de los Andes

Relaciones entre Entities

```
@Entity
public class Item {
    @Id
    protected Long itemId;
    protected String title;
    @OneToMany(mappedBy="item")
    protected Set<Bid> bids;
}

@Entity
public class Bid {
    @Id
    protected Long bidId;
    protected Double amount;
    @ManyToOne
    protected Item item;
}
```

Tomado de [4]

Relaciones Bidireccionales (8)

@ManyToMany

```
@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface ManyToMany {
    Class targetEntity() default void.class;
    CascadeType[] cascade() default {};
    FetchType fetch() default LAZY;
    String mappedBy() default "";
}
```

Universidad de los Andes

Relaciones entre Entities

```
@Entity
public class Category {
    @Id
    protected Long categoryId;
    protected String name;
    @ManyToMany
    protected Set<Item> items;
}

@Entity
public class Item {
    @Id
    protected Long itemId;
    protected String title;
    @ManyToMany(mappedBy="items")
    protected Set<Category> categories;
}
```

Tomado de [4]

Universidad de los Andes

Agenda

- Introducción
- Generalidades JPA
- **Entities (Entidades)**
  - Relaciones entre Entidades
  - **Mapeo a la Base de Datos**
  - Administración de Entidades
- Unidades de Persistencia
- Referencias

Universidad de los Andes

Mapeo a la Base de Datos

```
@Entity
@Table(name="CUSTOMER")
public class Customer implements Serializable {
   private Long id;

   @Column(name="FirstName", nullable=false)
   private String name;

   @Id
   @GeneratedValue
   public Long getId() {…}
}
```

CUSTOMER

Id	FirstName
PK	

Mapeo a la Base de Datos

- El mapeo de las relaciones a campos en la base de datos depende del tipo de relación
- Ejemplo de relaciones bidireccionales 1 a 1:

Empleado

codigo	nombre
PK	

Departamento

numero	jefe_codigo
PK	FK

Herencia

- Soporta tres (3) estrategias:
  - *Una tabla por cada clase de la jerarquía* → Buen soporte a las relaciones polimórficas entre *entidades* y queries que involucran la jerarquía de clases
  - *Una tabla por cada clase concreta* → Bajo soporte a las relaciones polimórficas en la jerarquía de clases. Requiere UNIONES SQL o queries separadas por cada subclase
  - *Subclases unidas* → Buen soporte a las relaciones polimórficas, pero requiere una o más operaciones de JOIN

```
public enum InheritanceType {
    SINGLE_TABLE,
    JOINED,
    TABLE_PER_CLASS
};
```

Universidad de los Andes

Agenda

- Introducción
- Generalidades JPA
- **Entities (Entidades)**
  - Relaciones entre Entidades
  - Mapeo a la Base de Datos
  - **Administración de Entidades**
- Unidades de Persistencia
- Referencias

Administración de Entidades

- Las entidades son administradas por el *EntityManager*
- El EntityManager es representado por referencias de la interfaz *javax.persistence.EntityManager*
- Cada instancia está asociada a un contexto persistente
- *Contexto Persistente* → Conjunto de entidades administradas/manejadas en ejecución que existen en un repositorio de datos específico
- La interfaz define métodos para interactuar con el contexto persistente

```
@PersistenceContext
EntityManager em;
```

EntityManager

- Ofrece servicios para:
  - Crear y remover instancias de una entidad
  - Buscar instancias por su llave primaria
  - Ejecutar consultas sobre las entidades
- Maneja el ciclo de vida de las entidades:
  - ***persist()*** → Inserta la instancia de una entidad en la BD
  - ***remove()*** → Borra la instancia de una entidad de la BD
  - ***merge()*** → Guarda en el contexto las modificaciones hechas en la entidad. Sincroniza el estado de entidades *detached.*
  - Otras funciones: ***flush()***, ***refresh()***, ***find()***

EntityManager (2)

- Similar en funcionalidad a:
  - Un *Session* en Hibernate Session
  - Un *PersistenceManager* en JDO (Java Data Objects)
- Fábrica de objetos de consulta
  - ***createQuery()*** → Crea un instancia de consulta para utilizar JPQL
  - ***createNamedQuery()*** → Crea una instancia predefinida de consulta
  - ***createNativeQuery()*** → Crea instancias de consultas para utilizar SQL nativo

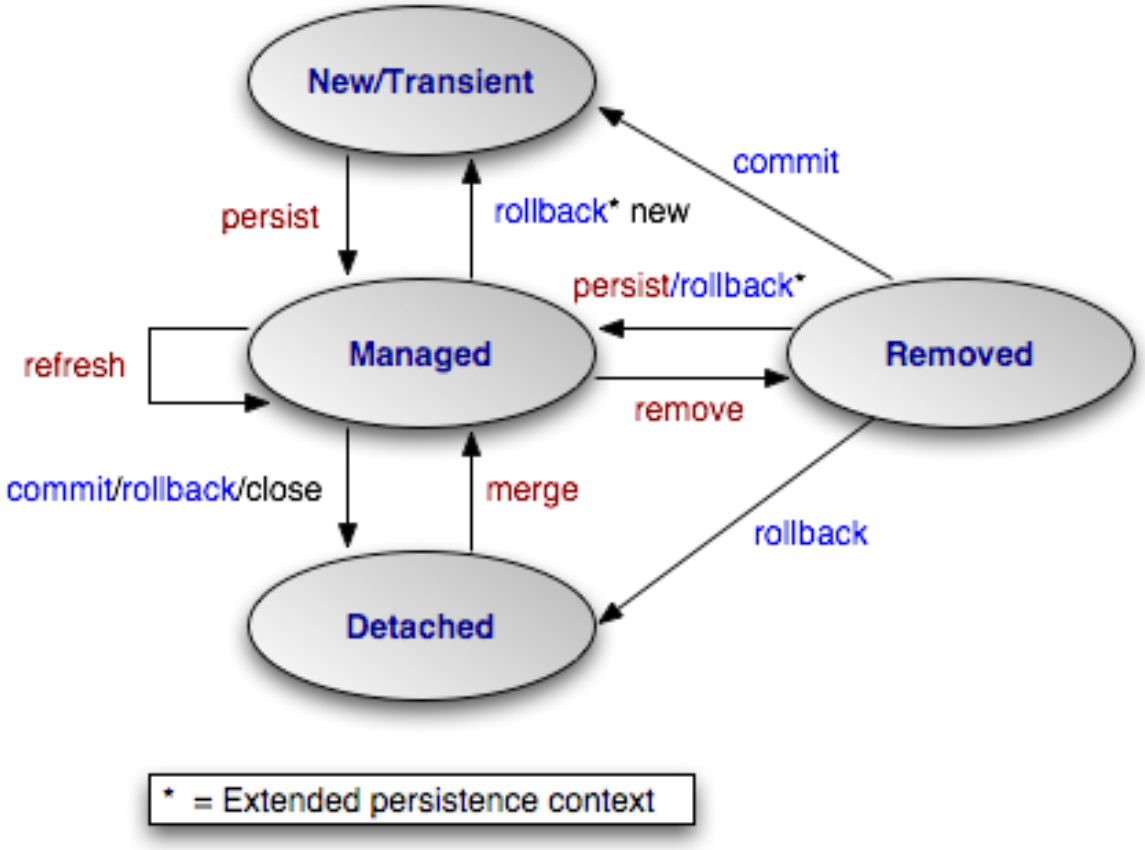
Ejemplos – Operaciones Persist, Find, Remove

```
public Order createNewOrder(Customer customer) {
    Order order = new Order(customer);
    entityManager.persist(order);
    return order;
}

public void removeOrder(Long orderId) {
    Order order = entityManager.find(Order.class, orderId);
    entityManager.remove(order);
}

public List findWithName(String name) {
    return em.createQuery("SELECT c FROM Customer c
        WHERE c.name LIKE :custName”)
        .setParameter("custName", name)
        .setMaxResults(10)
        .getResultList();
}
```

Ejemplo – Operación Merge

```
public OrderLine updateOrderLine(OrderLine orderLine)
{
    // The merge method returns a managed copy of
    // the given detached entity. Changes made to the
    // persistent state of the detached entity are
    // applied to this managed instance.
    return entityManager.merge(orderLine);
}
```

Ciclo de vida de entidades



Tomado de [3]

Ciclo de vida de entidades (2)

- **New Entity**
  - Creada utilizando el operador “new”
  - No tiene identidad persistente o estado
- **Managed Entity**
  - Tiene entidad persistente
  - Asociada con un contexto de persistencia
- **Detached Entity**
  - Tiene identidad persistente
  - No está asociada a un contexto de persistencia
- **Removed Entity**
  - Tiene identidad persistente
  - Asociada con un contexto de persistencia
  - Está programada para ser borrada del repositorio de datos

Universidad de los Andes

Administración de Entidades

- Si la instancia está asociada a otra instancia de entidad que invoca el método *persist* **y** si las relaciones de la entidad hacia la otra es *cascade=PERSIST* o *cascade=ALL*

- Entonces, al llamar al *EntityManager* para que persista

```
LineItem li = new LineItem(...);
order.getLineItems().add(li);
em.persist(li);
return li;
```

- El método *persist* es propagado a todas las entidades relacionadas al llamado de la entidad que tienen el elemento cascade en *ALL* o *PERSIST* en la anotación de la relación

Order.java

```
@OneToMany(cascade=CascadeType.ALL, mappedBy="order")
public Collection<LineItem> getLineItems()
{ return lineItems; }
```

Otro ejemplo – Modelo de Dominio

```
@Entity public class Department {
   @Id private int id;
   private String name;
   @OneToMany(mappedBy="dept", fetch=LAZY)
   private Collection<Employee> emps = new ...;
          ...
}
```

```
@Entity public class Employee {
   @Id private int id;
   private String firstName;
   private String lastName;
   @ManyToOne(fetch=LAZY)
   private Department dept;
       ...
}
```

Otro ejemplo – Manejo de Relaciones (2)

```
public int addNewEmployee(...) {
   Employee e = new Employee(...);
   Department d = new Department(1, ...);

   e.setDepartment(d);
   //Reverse relationship is not set
   em.persist(e);
   em.persist(d);

   return d.getEmployees().size();
}
```

INCORRECTO

Otro ejemplo – Manejo de Relaciones (3)

```
public int addNewEmployee(...) {
   Employee e = new Employee(...);
   Department d = new Department(1, ...);

   e.setDepartment(d);
   d.getEmployees().add(e);
   em.persist(e);
   em.persist(d);

   return d.getEmployees().size();
}
```

CORRECTO

Consulta de Entidades

createQuery

```
em.createQuery("SELECT c FROM  Customer c WHERE c.name
        LIKE :custName")
   .setParameter("custName", name)
   .setMaxResults(10)
   .getResultList();
```

createNamedQuery

- Declaración

```
@NamedQueries(
{    @NamedQuery(
        name="findAllCustomersWithName",
        query="SELECT c FROM Customer c WHERE c.name
             LIKE :custName" )
}
```

- Uso

```
em.createNamedQuery("findAllCustomersWithName")
     .setParameter("custName", "Smith").getResultList();
```

Tipos de EntityManager

- *Container-Managed EntityManager* → Mundo *Java EE*
  - El ciclo de vida del *EntityManager* es manejado por el contenedor.
  - El contexto es propagado automáticamente a todos los componentes de la aplicación que usan la instancia en una transacción
  - Inyección:

```
@PersistenceContext
EntityManager em;
```

Tipos de EntityManager (2)

- *Application-Managed EntityManager* → Mundo *Java SE*
  - El ciclo de vida del *EntityManager* es manejado por la aplicación
  - El contexto NO es propagado a los otros componentes
  - El desarrollador se encarga de manejo transaccional
  - Inyección:

```
@PersistenceUnit
private EntityManagerFactory emf;
EntityManager em = emf.createEntityManager();

@Resource
private UserTransaction utx;
```

Universidad de los Andes

Agenda

- Introducción
- Generalidades JPA
- Entities (Entidades)
  - Relaciones entre Entidades
  - Mapeo a la Base de Datos
  - Administración de Entidades
- **Unidades de Persistencia**
- Referencias

Universidad de los Andes

Unidades de Persistencia

- Conjunto de todas las clases administradas por el *EntityManager* en la aplicación
- Definidas en el archivo de configuración *persistence.xml*
- Pueden ser empaquetadas como parte de un EJB JAR file, o como un JAR incluido en un WAR o EAR
- Estas unidades son copiadas al contenedor para ser desplegadas (deployed)

Unidades de Persistencia

persistence.xml

- El archivo *persistence.xml* define una o más unidades de persistencia
- Cada unidad de persistencia depende de un *DataSouce*

```
<persistence>
    <persistence-unit name="OrderManagement">
        <description>This unit manages orders and customers.
            It does not rely on any vendor-specific features and can
            therefore be deployed to any persistence provider.
        </description>
        <jta-data-source>jdbc/MyOrderDB</jta-data-source>
        <jar-file>MyOrderApp.jar</jar-file>
        <class>com.widgets.Order</class>
        <class>com.widgets.Customer</class>
    </persistence-unit>
</persistence>
```

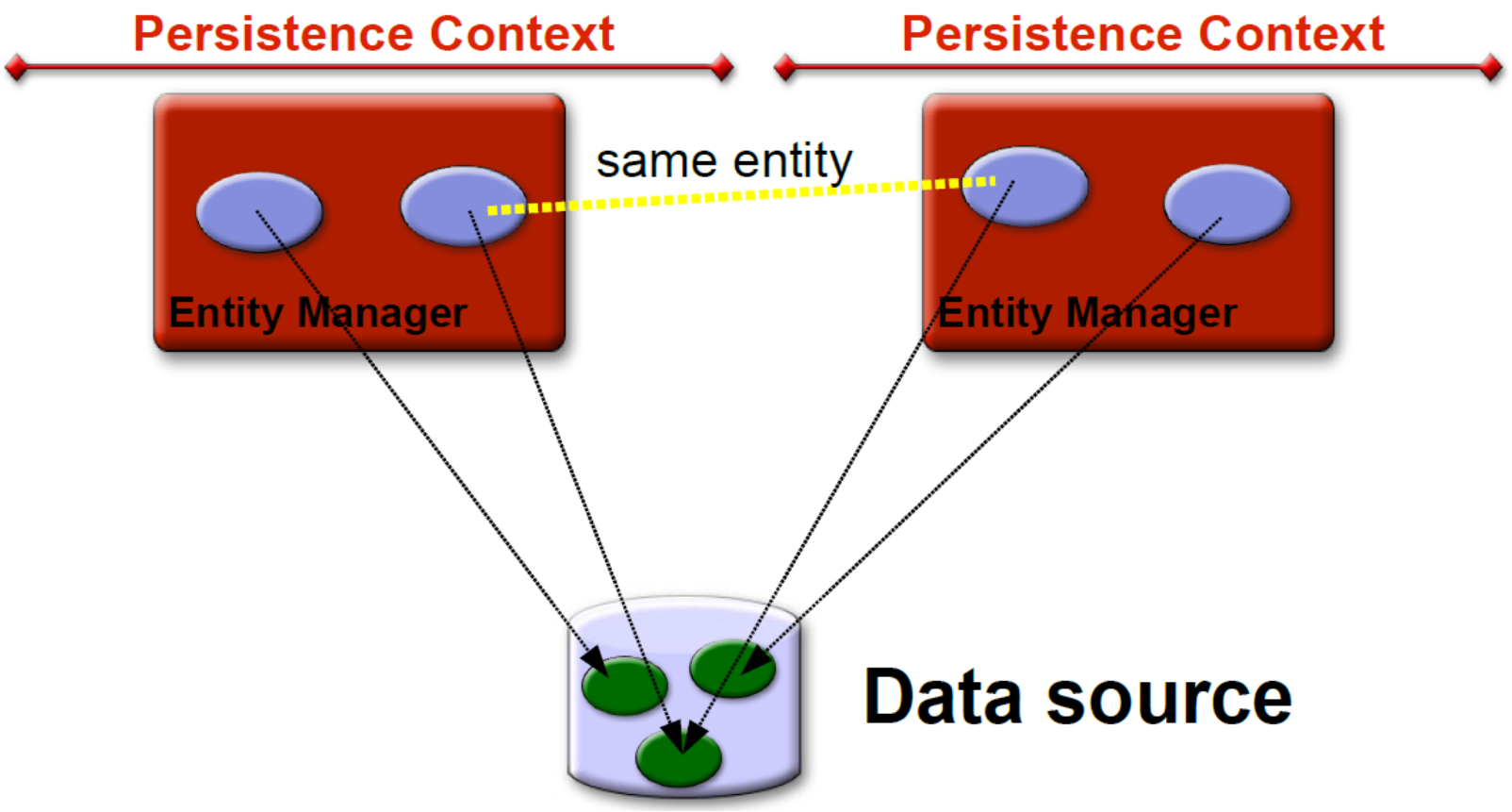
DataSource

- Un *DataSource* define un pool de conexiones a la BD
- Se pueden configurar desde la consola de administración del servidor de aplicaciones

```
<datasources>
    <local-tx-datasource>
        <jndi-name>MyOrderDB</jndi-name>
        <connection-url>jdbc:oracle:thin:@chie.uniandes.edu.co:1522:chie10</connection-url>
        <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
        <user-name>------</user-name>
        <password>-----</password>
    </local-tx-datasource>
</datasources>
```

Universidad de los Andes

Unidades de Persistencia



Universidad de los Andes

Agenda

- Introducción
- Generalidades JPA
- Entities (Entidades)
  - Relaciones entre Entidades
  - Mapeo a la Base de Datos
  - Administración de Entidades
- Unidades de Persistencia
- **Referencias**

Referencias

1. **Java EE Enterprise Application Technologies.** http://www.oracle.com/technetwork/java/javaee/tech/entapps-138775.html
2. **Java Persistence API Architecture.** http://openjpa.apache.org/builds/1.0.2/apache-openjpa-1.0.2/docs/manual/jpa_overview_arch.html
3. **Entity Lifecycle Management** http://openjpa.apache.org/builds/1.0.2/apache-openjpa-1.0.2/docs/manual/jpa_overview_em_lifecycle.html
4. **The Java™ EE 6 Tutorial.** Eric Jendrock. Oracle Corporation. 2011.
5. **EJB 3 in Action.** Panda Debu, Rahman Reza, Lane Derek. Manning. 2007.
6. **EJB 3 Developer Guide.** Sikora, Michael. 2008.

Universidad de los Andes

Gracias !!!

Rafael Meneses

rg.meneses81@uniandes.edu.co