Tutoriales Curso JPA

<?xml version="1.0"?><DocumentFileOSQ xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"> <GraphicCharterDefinitionId>0</GraphicCharterDefinitionId> <TemplateBaseTypeId>0</TemplateBaseTypeId> <CompanyId>1</CompanyId> <ConfidentialId>0</ConfidentialId> <ConfidentialDescription /> <CountryId>20</CountryId> <PageSizeId>1</PageSizeId> <PageOrientationId>1</PageOrientationId> <PrePrintedStationary>false</PrePrintedStationary> <Project>GEMIS</Project> <Reference>20170124-170150-esgomez</Reference> <TemplateType>3</TemplateType> <CultureId>fr-FR</CultureId> <LanguageId>2</LanguageId> <Customer>Centro de Servicio Valencia</Customer> <DocumentDate>2018-12-12T14:16:38.6304655+01:00</DocumentDate> <Saved>true</Saved> <IsValid>true</IsValid> <FirstPageCover>false</FirstPageCover> <IsNew>false</IsNew> <CurrentVersion>1.00</CurrentVersion> <DocumentType>Solution Requirements Specification</DocumentType> <DocumentTypeId>-1</DocumentTypeId> <Entity /> <HasDistributionList>false</HasDistributionList> <HasForeword>false</HasForeword> <Recipient>Customer Stakeholders (including at least the IT PM and the Stakeholder and User manager)</Recipient> <Title>Spécifications des besoins de la solution - GEMIS</Title> <Status>2</Status> <StatusDescription>Projet</StatusDescription> <SetEdition>false</SetEdition> <SetVersion>false</SetVersion> <TemplateEditor>EN\_eMREQ-SRS</TemplateEditor> <TemplateVersionMajor>1.2</TemplateVersionMajor> <TemplateVersionMinor>4</TemplateVersionMinor> <TemplateYear>2016</TemplateYear> <TemplateState>1</TemplateState></DocumentFileOSQ>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | |  | |  |
|  |  |  | |  | |  |
|  |  | Formación Sopra | | | |  |
|  |  |  |
|  |  | JPA básico | | | |  |
|  |  | 1.0 Lunes 21 Noviembre 2018 | | | |  |
|  |  |
|  |  |  | | |  |  |

Historial

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Versión** | **Fecha** | **Origen de la actualización** | **Redactado por** | **Validado por** |
| 1.0 | 26/11/2018 | Primera versión del documento | Alba Bermejo Solís  Adrián Colmena Mateos  Emilio Guillem Simón |  |
|  | | | | | |

Contenidos

[1. Tutorial Capítulo 2 4](#_Toc532387530)

[1.1. Configuración inicial 4](#_Toc532387531)

[1.2. Creación de una entidad y conexión 7](#_Toc532387532)

[1.3. Creación de servicios de entidad 11](#_Toc532387533)

[1.4. El método main() 13](#_Toc532387534)

[2. Tutorial Capítulo 3 14](#_Toc532387535)

[2.1. Creación de entidades 14](#_Toc532387536)

[2.2. Generación de identificadores 19](#_Toc532387537)

[2.3. Creación de objeto embebido 20](#_Toc532387538)

[2.4. Método main() para comprobar funcionamiento 25](#_Toc532387539)

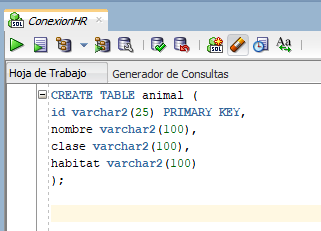
# Tutorial Capítulo 2

A continuación, se va a introducir a la parte práctica de desarrollar una aplicación Java con JPA. Para ello lo primero será configurar nuestro IDE de programación (Eclipse) para que pueda conectar con nuestra base de datos, en este caso la de Oracle (SQL Developer, HR).

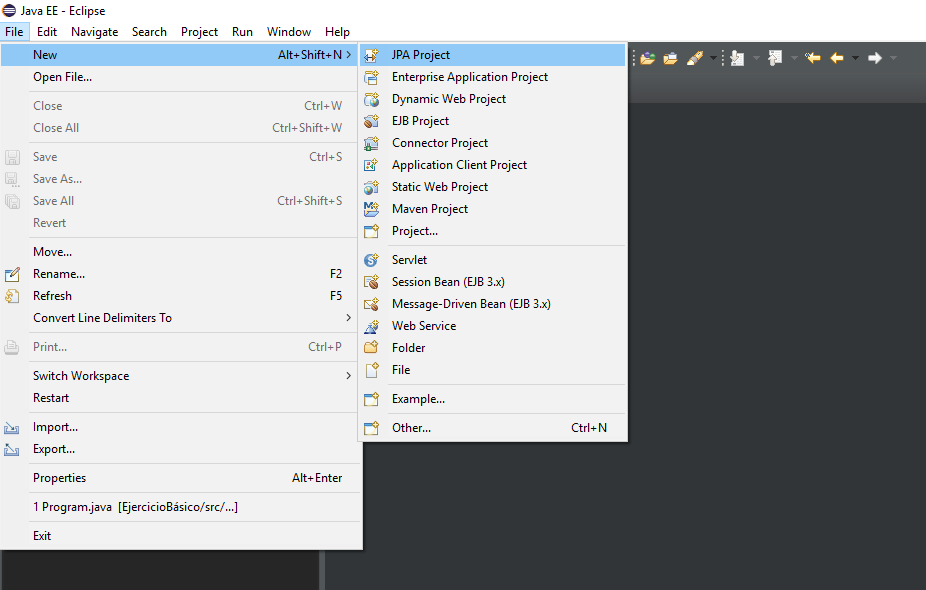
## Configuración inicial

En primer lugar, es necesaria una base de datos (usaremos HR de Oracle) y crear las tablas sobre las que se vaya a trabajar. En este tutorial, solo se va a utilizar una tabla.

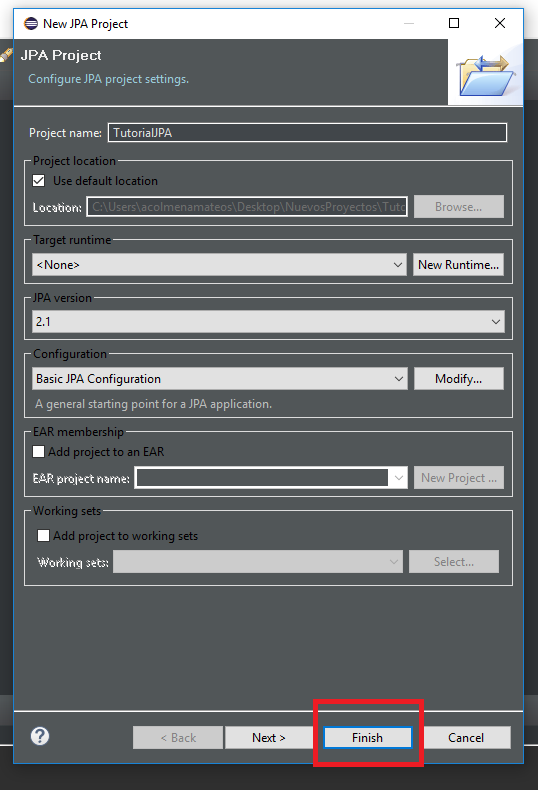
A continuación, se crea la tabla *animal* en SQL Developer con los campos id (clave primaria), nombre del animal (en latín), clase y hábitat. Todos los campos serán de tipo varchar2.



Una vez creada la tabla, será necesario conectarla con el IDE Eclipse. En Eclipse, se crea un nuevo proyecto JPA con su archivo Persistence.xml.







Para que el proyecto pueda conectar con base de datos es necesario un driver, en este caso usaremos el de Oracle **ojdbc6.jar**, pudiendo descargarlo en la siguiente URL:

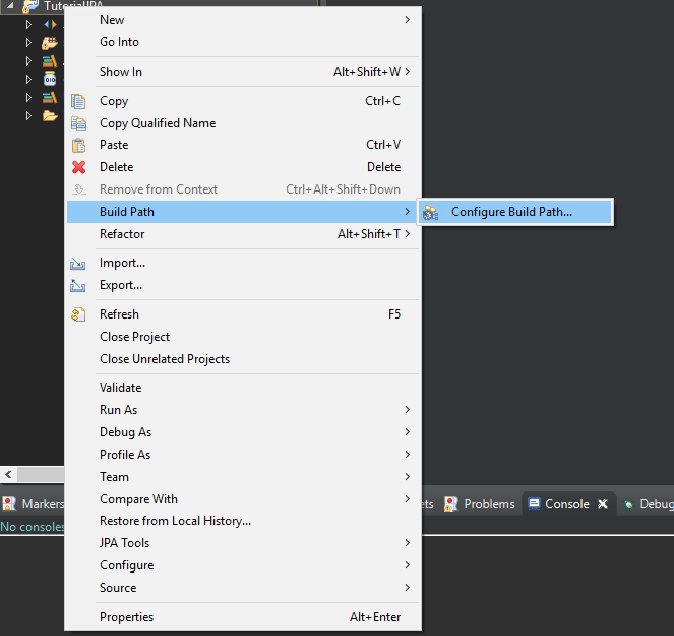
<http://www.java2s.com/Code/Jar/o/Downloadojdbc6jar.htm>

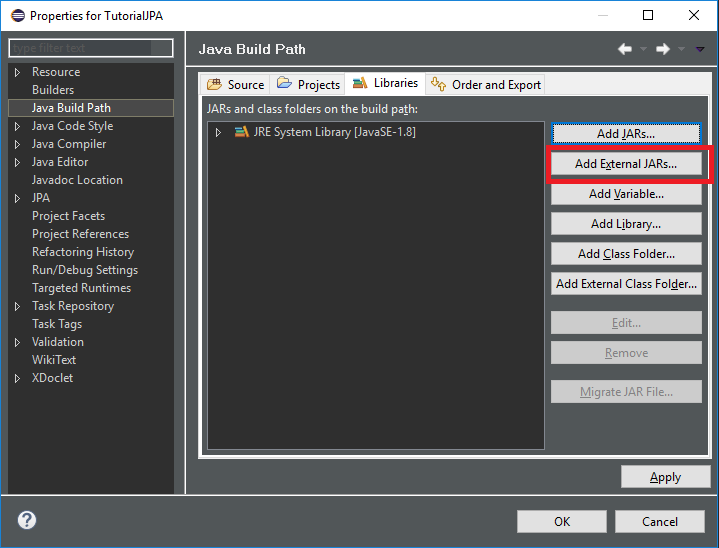
Y descargar las librerías de Hibernate, en esta página:

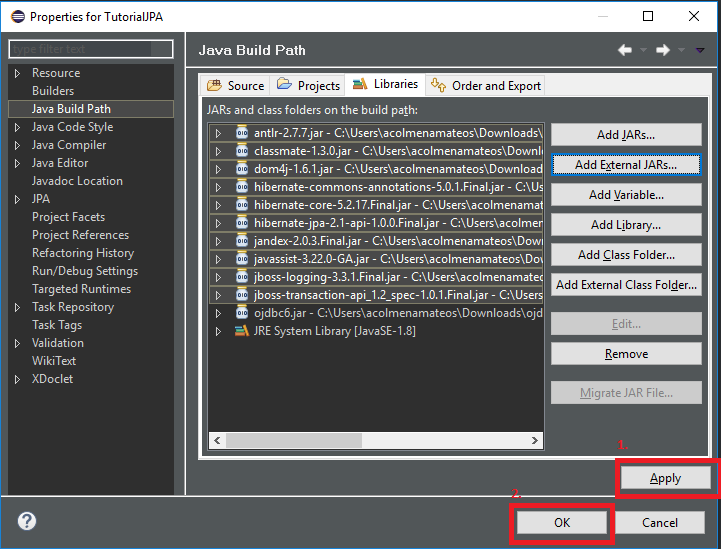
<https://sourceforge.net/projects/hibernate/files/hibernate-orm/5.2.17.Final/hibernate-release-5.2.17.Final.zip/download>

Hibernate funciona como el *persistence provider* del proyecto JPA creado. No es necesario definirlo en el persistance.xml ya que lo detecta de manera automática una vez que las librerías sean añadidas al proyecto.

Para añadir las librerías de Hibernate, así como el driver de Oracle, es necesario configurar el *path* del proyecto.







Destacar, que las librerías de Hibernate, se encuentran en lib->required del Zip descargado anteriormente.

## Creación de una entidad y conexión

Para crear una entidad, es necesario crear un paquete (com.JPAmanual.entidad) y una clase java común (Animal) con la anotación *@Entity*, y la anotación *@Id* para su clave primaria. Es necesario definir los campos de la tabla, a la que hace referencia la entidad, como propiedades de la clase. Añadir también su constructor, un constructor vacío y los métodos *getters* y *setters* de todas las propiedades. La entidad necesita implementar la interfaz *Serializable,* que obliga a sobrescribir (*@Override*) los métodos *equals, hashCode y toString.*

**package** com.JPAmanual.entidad;

**import** java.io.Serializable;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.NamedQueries;

**import** javax.persistence.NamedQuery;

@Entity

**public** **class** Animal **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@Id

**private** String id;

**private** String nombre;

**private** String clase;

**private** String hábitat;

**public** Animal() {

}

**public** Animal(String id, String nombre, String clase, String hábitat) {

**this**.id = id;

**this**.nombre = nombre;

**this**.clase = clase;

**this**.hábitat = hábitat;

}

**public** String getId() {

**return** id;

}

**public** **void** setId(String id) {

**this**.id = id;

}

**public** String getNombre() {

**return** nombre;

}

**public** **void** setNombre(String nombre) {

**this**.nombre = nombre;

}

**public** String getClase() {

**return** clase;

}

**public** **void** setClase(String clase) {

**this**.clase = clase;

}

**public** String getHábitat() {

**return** hábitat;

}

**public** **void** setHábitat(String hábitat) {

**this**.hábitat = hábitat;

}

@Override

**public** **boolean** equals(Object elOtro) {

**if** (elOtro **instanceof** Animal) {

Animal a = (Animal) elOtro;

**return** **this**.id == a.id;

}

**return** **false**;

}

@Override

**public** **int** hashCode() {

**return** **this**.id.hashCode();

}

@Override

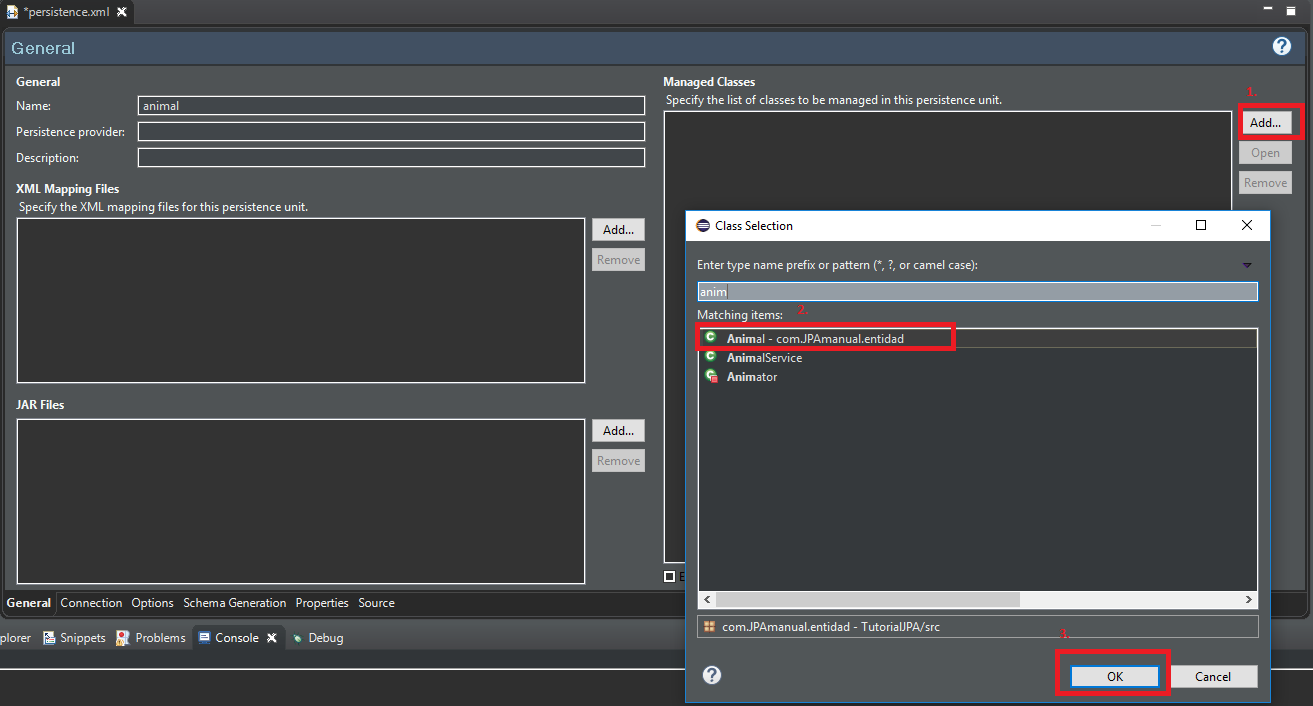
**public** String toString() {

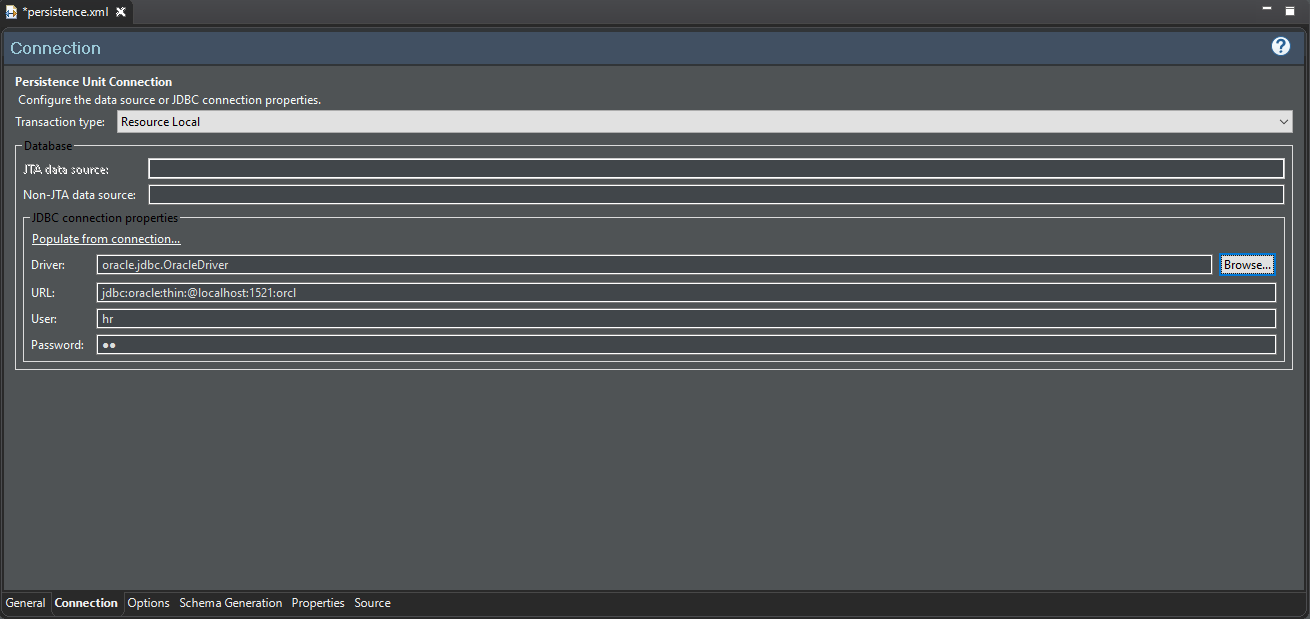
**return** String.*format*("id=%s, nombre=%s, clase=%s, hábitat=%s", **this**.id,   
 **this**.nombre, **this**.clase, **this**.hábitat);

}

}

Por último, es necesario configurar el archivo persistence.xml del proyecto. Dicho archivo debe contener las entidades y la configuración del driver.





Una vez configurado los parámetros anteriores, el código del archivo persistence.xml queda de la siguiente manera:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<persistence version=*"2.1"*

xmlns=*"http://xmlns.jcp.org/xml/ns/persistence"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence\_2\_1.xsd"*>

<persistence-unit name=*"animal"* transaction-type=*"RESOURCE\_LOCAL"*>

<class>com.JPAmanual.entidad.Animal</class>

<properties>

<property name=*"javax.persistence.jdbc.url"* value=*"jdbc:oracle:thin:@localhost:1521:orcl"* />

<property name=*"javax.persistence.jdbc.user"* value=*"hr"* />

<property name=*"javax.persistence.jdbc.password"* value=*"hr"* />

<property name=*"javax.persistence.jdbc.driver"* value=*"oracle.jdbc.OracleDriver"* />

</properties>

</persistence-unit>

</persistence>

## Creación de servicios de entidad

Se crea un nuevo paquete (com.JPAmanual.servicios) y una nueva clase (ServicioAnimal), donse se crearán los métodos CRUD.

* Método crearAnimal.

Para crear un animal, es necesario pasarle como parámetros el *EntityManager* (em), y cada una de las propiedades de la clase Animal. Dentro del método creamos un objeto Animal, y luego se lo pasamos al *EntityManager* para que ejecute el método *persist()*.

**package** com.JPAmanual.servicios;

**import** java.util.ArrayList;

**import** java.util.List;

**import** javax.persistence.EntityManager;

**import** com.JPAmanual.entidad.Animal;

**public** **class** ServicioAnimal {

// Persist

**public** **static** Animal crearAnimal(EntityManager em, String id, String nombre,   
 String clase, String hábitat) {

Animal a = **new** Animal(id, nombre, clase, hábitat);

em.persist(a);

**return** a;

}

* Método encontrarAnimal.

Para poder encontrar un animal en concreto de la base de datos y que nos lo devuelva como un objeto de Java se le pasa por parámetros el *EntityManager* (em), y la clave primaria (id). El *EntityManager* localizará al animal con el método *find()*.

// Find

**public** **static** Animal encontrarAnimal(EntityManager em, String id) {

**return** em.find(Animal.**class**, id);

}

* Método borrarAnimal.

Si se quiere borrar un registro de la base de datos desde Java será necesario otra vez pasarle el *EntityManager* (em), y la clave primaria (id), para que encuentre el animal que se quiere borrar. Una que el *EntityManager* lo encuentre, ejecuta el comando *remove()*. Como condición para poder eliminarlo debe haber encontrado un registro con valor disitinto a *null.* Este método devolverá *true* si ha eliminado algún registro de la base de datos y *false*, si el registro no existe o se ha identificado de manera incorrecta.

// Remove

**public** **static** **boolean** borrarAnimal(EntityManager em, String id) {

Animal a = em.find(Animal.**class**, id);

**if** (a != **null**) {

em.remove(a);

**return** **true**;

}

**return** **false**;

}

* Método modificarAnimal.

Si se quiere actualizar un registro de la base de datos desde Java será necesario otra vez pasarle el *EntityManager* (em), la clave primaria (id), para que encuentre el animal que se quiere modificar. Además, se le debe pasar el resto de propiedades. Una vez que el *EntityManager* lo encuentre, se usan los métodos *set()* de aquellos campos que se quiera cambiar el valor*.* Este método devolveráel objeto modificado.

// Update Animal

**public** **static** Animal modificarAnimal(EntityManager em, String id, String nombre,   
 String clase, String hábitat) {

Animal a = em.find(Animal.**class**, id);

**if** (a != **null**) {

a.setNombre(nombre);

a.setClase(clase);

a.setHábitat(hábitat);

}

**return** a;

}

* Método getNombres.

Si se quiere obtener una lista de los valores de un campo, se utilizan *queries*. Para ello utilizamos la anotación *@NamedQueries*, y la anotación *@NamedQuery* para cada una de las búsquedas que se quieran hacer, en el código de la entidad. Estas *queries* se escribirán en lenguaje JPQL.

**import** javax.persistence.NamedQueries;

**import** javax.persistence.NamedQuery;

@Entity

@NamedQueries({ @NamedQuery(name = "Animal.nombre", query = "SELECT a.nombre FROM Animal a"), })

**public** **class** Animal **implements** Serializable {

Luego en la clase servicios, se incluirá el método que ejecutará esa *query* en la base de datos y nos devolverá el resultado como un objeto *List.* A dicho método se le debe pasar el *EntityManager* como parámetro, el cual utiliza el método *createNamedQuery(“*Nombre de la query”, Clase de la propiedad *).getResultList().*

**public** **static** List<String> getNombres(EntityManager em) {

List<String> Nombres = **new** ArrayList<String>();

Nombres = em.createNamedQuery("Animal.nombre", String.**class**).getResultList();

**return** Nombres;

}

## El método main()

Se crea una última clase (Programa) dentro de un nuevo paquete (com.JPAmanual.main).

Este método será el encargado de ejecutar todos los métodos vistos en la sección anterior. Para ello, se deben crear la *EntityManagerFactory* y los *EntityManger* que se requieran. El objeto *EntityManagerFactory* debe contener el nombre de persistence unit definida en el fichero *persistence.xml*, a la cual se le ha dado el nombre de la tabla en base de datos (animal).

Además, para la correcta ejecución de los métodos CRUD, es necesario abrir una transacción antes de la ejecución de estos. Una vez ejecutados todos los métodos se deberá hacer el *commit* para que los cambios se reflejen en base de datos y la transacción se cierre y cerrar los *EntityManager* y la *EntityManagerFactory.*

**package** com.JPAmanual.main;

**import** javax.persistence.EntityManager;

**import** javax.persistence.EntityManagerFactory;

**import** javax.persistence.Persistence;

**import** com.JPAmanual.entidad.Animal;

**import** com.JPAmanual.servicios.ServicioAnimal;

**public** **class** Programa {

**public** **static** **void** main(String[] args){

EntityManagerFactory emf = Persistence.*createEntityManagerFactory*("animal");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

Animal Hipo = ServicioAnimal.*crearAnimal*(em, "Hippo", "Hippopotamus amphibius",   
 "Mamífero", "Sabana");

Animal encontrado= ServicioAnimal.*encontrarAnimal*(em, "Lion");

System.***out***.println(encontrado);

**if**(ServicioAnimal.*borrarAnimal*(em, "Sparrow"))

System.***out***.println("Animal eliminado”);

**else**

System.***out***.println("Error al eliminar");

ServicioAnimal.*modificarAnimal*(em, "Hippo", "Hipopótamo", "Herbívoro", "África");

System.***out***.println(ServicioAnimal.*getNombres*(em));

em.getTransaction().commit();

em.close();

emf.close();

}

}

# Tutorial Capítulo 3

En este tutorial se construirá un pequeño proyecto para revisar el contenido visto en el capítulo 3. Para ello, se utilizará la base de datos *hr* de *Oracle*.

En primer lugar, se crearán, con ayuda de las anotaciones vistas en el capítulo, dos entidades relacionadas: *Employee* y *Departments*.

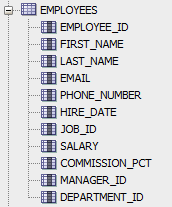
Por último, se creará una tercera clase que servirá como objeto embebido común a ambas entidades.

**Nota:** Se da por sabido cómo crear un proyecto JPA y conectar con la base de datos de Oracle.

## Creación de entidades

Antes de nada, es necesario reflexionar qué tipo de relación existe entre ambas entidades. Se considerará que cada empleado pertenece a un departamento y que en un departamento hay varios empleados. Por lo tanto, visto desde *Department* hacia *Employee*, se trataría de una relación *OneToMany*; mientras que visot desde *Employee* hacia *Department* se trataría de una relación *ManytoOne*.

Para crear las entidades *Employee* y *Departments* de acuerdo a las tablas *Employees* y *Departments* de la base de datos ya existente, se deberá, en primer lugar, consultar la misma y decidir qué columnas se querrán representar en la entidad. Esto quiere decir que no es necesario que todas las columnas de una tabla ya creada tengan que ser consideradas en la creación de una entidad, ya que es posible que no se quiera trabajar en Java con todos los atributos, sino con una selección de ellos. Siempre hay que tener en cuenta que en la base de datos existen columnas que no pueden ser nulas, por lo que deben estar en Java si se quieren crear nuevos registros.



Para la creación de la entidad *Employee* se decide tener en cuenta las siguientes columnas de la tabla: *EMPLOYEE\_ID* (que, evidentemente, será el identificador), *FIRST\_NAME, LAST\_NAME, EMAIL* (se pretende que esta información no se cargue hasta que no sea referenciada)*, PHONE\_NUMBER,* *JOB\_ID, HIRE\_DATE* (del paquete *util*, no del paquete *sql*)*, SALARY* y *DEPARTMENT\_ID.* Esta última columna sirve únicamente para relacionar ambas entidades, sin embargo, en Java, esta relación se representa de una forma diferente. Por lo tanto, la clase *Employee* no contendrá el id del departamento, sino un objeto de la clase *Department* con todos sus atributos.

Asimismo, se les proporciona un nombre a los atributos de la clase y, gracias a la anotación *@Column,* se les asocia con la columna correspondiente de la siguiente manera:

**package** com.sopra.tutorialchapter3.entity;

**import** java.io.Serializable;

**import** java.util.Date;

**import** javax.persistence.Basic;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.FetchType;

**import** javax.persistence.Id;

**import** javax.persistence.ManyToOne;

**import** javax.persistence.Table;

**import** javax.persistence.Temporal;

**import** javax.persistence.TemporalType;

**import** javax.persistence.JoinColumn;

@Entity

@Table (name="EMPLOYEES")

**public class** Employee **implements** Serializable {

@Id

@Column(name= "EMPLOYEE\_ID")

**private** **long** id;

@Column(name="FIRST\_NAME")

**private** String name;

@Column (name="LAST\_NAME")

**private** String surname;

@Basic(fetch=FetchType.***LAZY***)

@Column(name="EMAIL")

**private** String email;

@Column(name="PHONE\_NUMBER")

**private** String phone;

@Column (name="JOB\_ID")

**private** String jobId;

@Column(name="HIRE\_DATE")

@Temporal(TemporalType.***DATE***)

**private** Date hireDate;

@Column(name="SALARY")

**private** **long** salary;

@ManyToOne

@JoinColumn(name="DEPARTMENT\_ID")

**private** Department department;

**public** Employee(){}

**public** Employee(**long** id, String name, String phone, **long** salary, String email,

String surname, Department department, Date hireDate, String jobId) {

**this**.id = id;

**this**.name = name;

**this**.phone = phone;

**this**.hireDate = hireDate;

**this**.salary = salary;

**this**.email = email;

**this**.surname = surname;

**this**.department = department;

**this**.jobId = jobId;

}

**public** **long** getId() {

**return** id;

}

**public** void setId(**long** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** void setName(String name) {

**this**.name = name;

}

**public** String getPhone() {

**return** phone;

}

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

**public** Date getHireDate() {

**return** hireDate;

}

**public** **void** setHireDate (Date hireDate) {

**this**.hireDate = hireDate;

}

**public** **long** getSalary() {

**return** salary;

}

**public** **void** setSalary(**long** salary) {

**this**.salary = salary;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getSurname() {

**return** surname;

}

**public** **void** setSurname(String surname) {

**this**.surname = surname;

}

**public** Department getDepartment() {

**return** department;

}

**public** **void** setDepartment(Department department) {

**this**.department = department;

}

**public** String getJobId () {

**return** jobId;

}

**public** **void** setJobId (String jobId) {

**this**.jobId = jobId;

}

@Override

**public** String toString() {

**return** String.*format*("id=%d, nombre=%s, apellido=%s, salario=%d,

departamento=%d ", **this**.id, **this**.name, **this**.surname, **this**.salary,

**this**.department.getId());

}

@Override

**public** **int** hashCode() {

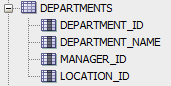
**return** (**int**)**this**.id;

}

}

Se puede observer la manera en la que se utilizan las anotaciones *@Entity*, *@Table* (indicando el nombre de la tabla en la base de datos), *@Id*, *@ManytoOne* y *@Basic(fetch=FetchType.LAZY)*. Además, como en toda entidad de JPA, se ha declarado un contructor vacío del que hará uso el proveedor de persistencia. También, para uso del desarrollador, se ha declarado otro constructor no vacío. Por ultimo, ha sido necesario declarar todos los métodos g*etters* y *setters*.

De igual modo se procederá para la entidad *Department,* para la que se elegirán las columnas: *DEPARTMENT\_ID* y *DEPARTMENT\_NAME.* Además, se verá que no es necesario el uso de la anotación *@Column* si se nombra al atributo igual que la columna a la que hace referencia. Se usarán los nombres dados en los campos (en lugar de en las propiedades, es decir, los métodos *getter*), para ello se empleará la anotaicón *@Access(AccessType.FIELD)*. Sin embargo, para poner en práctica el acceso mixto, al nombre del departamento, si su longitud es menor de 10 caracteres, se le añadirá “dep. ” al principio y si no, se quedará con el nombre original. Esto se hará mediante el uso de las anotaciones *@Access(AccessType.PROPERTY)* junto con la de *@Column,* y la de *@Transient*.



**package** com.sopra.tutorialchapter3.entity;

**import** java.io.Serializable;

**import** java.util.List;

**import** javax.persistence.Access;

**import** javax.persistence.AccessType;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.OneToMany;

**import** javax.persistence.Table;

**import** javax.persistence.Transient;

@Entity

@Table(name="Departments")

@Access(AccessType.***FIELD***)

**public** **class** Department **implements** Serializable{

@Id

**private** **long** department\_id;

@Transient

**private** String department\_name;

@OneToMany(mappedBy="department")

**private** List<Employee> employee;

**public** Department() {

}

**public** Department(**long** department\_id, String department\_name, List<Employee>

employee) {

**this**.department\_id = department\_id;

**this**.department\_name = department\_name;

**this**.employee = employee;

}

**public** **long** getId() {

**return** department\_id;

}

**public** **void** setId(**long** department\_id) {

**this**.department\_id = department\_id;

}

**public** String getName() {

**return** department\_name;

}

**public** **void** setName(String department\_name) {

**this**. department\_name = department\_name;

}

@Access(AccessType.***PROPERTY***) @Column(name="DEPARTMENT\_NAME")

**public** String getNameWithDep (){

**if**(department\_name.length()<10) **return** "dep. " + department\_name;

**else** **return** department\_name;

}

**public** **void** setNameWithDep(String name){

**if**(name.contains("dep. ")) **this**.department\_name= name.substring(5);

**else** **this**.department\_name=name;

}

**public** List<Employee> getEmployee() {

**return** employee;

}

**public** **void** setEmployee(List<Employee> employee) {

**this**.employee = employee;

}

@Override

**public** String toString() {

**return** String.*format*("id=%d, nombre=%s”,

**this**. department\_id, **this**.department\_name);

}

@Override

**public** **int** hashCode() {

**return** (**int**)**this**.department\_id;

}

}

Se puede observar como en esta entidad, para crear correctamente la relación bidireccional que se ha planteado anteriormente, se declara el atributo *employee* de tipo lista de *Employee* con la anotación *@OneToMany(mappedBy="department").* Esta indica que esta entidad *Department* está en la parte inversa de la relación, es decir, que no es la propietaria de la relación y que en cada departamento hay varios empleados.

## Generación de identificadores

A continuación se utilizará la generación de identificadores mediante secuencias para ambas entidades. Para ello, será necesaria la creación de una secuancia en Oracle para cada entidad:

CREATE SEQUENCE Emp\_Seq

MINVALUE 1

START WITH 1

INCREMENT BY 1;

CREATE SEQUENCE Dep\_Seq

MINVALUE 1

START WITH 1

INCREMENT BY 1;

//. . .

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.SequenceGenerator;

@Entity

@Table (name="EMPLOYEES")

**public class** Employee **implements** Serializable {

@SequenceGenerator(name="Emp\_Gen", sequenceName="Emp\_Seq")

@Id @GeneratedValue(generator="Emp\_Gen")

@Column(name= "EMPLOYEE\_ID")

**private** **long** id;

//. . .

}

//. . .

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.SequenceGenerator;

@Entity

@Table(name="DEPARTMENTS")

**public** **class** Department **implements** Serializable{

@SequenceGenerator(name="Dep\_Gen", sequenceName="Dep\_Seq")

@Id @GeneratedValue(generator="Dep\_Gen")

@Column(name="DEPARTMENT\_ID")

**private** **long** department\_id;

//. . .

}

## Creación de objeto embebido

A continuación, se creará un objeto embebido aprovechando que tanto la tabla *Departments* como la tabla *Employees* tienen la columna *MANAGER\_ID*. Por ello, se añade en ambas entidades esta propiedad (con la anotación *@Embedded*) así como sus métodos *getters* y *setters*,y se actualiza el constructor de la clase. También se crea una clase nueva para *Manager* que será anotada como *@Embeddable*. Las clases *Employee* y *Department* quedarían de la siguiente manera:

**package** com.sopra.tutorialchapter3.entity;

**import** java.io.Serializable;

**import** java.util.Date;

**import** javax.persistence.Basic;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.FetchType;

**import** javax.persistence.Id;

**import** javax.persistence.ManyToOne;

**import** javax.persistence.Table;

**import** javax.persistence.Temporal;

**import** javax.persistence.Column;

**import** javax.persistence.TemporalType;

**import** javax.persistence.Embedded;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.SequenceGenerator;

**import** com.sopra.tutorialchapter3.embededobject.Manager;

@Entity

@Table (name="EMPLOYEES")

**public class** Employee **implements** Serializable {

@SequenceGenerator(name="Emp\_Gen", sequenceName="Emp\_Seq")

@Id @GeneratedValue(generator="Emp\_Gen")

@Column(name= "EMPLOYEE\_ID")

**private** **long** id;

@Column(name="FIRST\_NAME")

**private** String name;

@Column (name="LAST\_NAME")

**private** String surname;

@Basic(fetch=FetchType.***LAZY***)

@Column(name="EMAIL")

**private** String email;

@Column(name="PHONE\_NUMBER")

**private** String phone;

@Column (name="JOB\_ID")

**private** String jobId;

@Column(name="HIRE\_DATE")

@Temporal(TemporalType.***DATE***)

**private** Date hireDate;

@Column(name="SALARY")

**private** **long** salary;

@ManyToOne

@JoinColumn(name="DEPARTMENT\_ID")

**private** Department department;

@Column(name= "MANAGER\_ID")

@Embedded

**private** Manager managerId;

**public** Employee(){}

**public** Employee(**long** id, String name, String phone, **long** salary, String email,

String surname, Department department, Manager managerId, Date hireDate,

String jobId) {

**this**.id = id;

**this**.name = name;

**this**.phone = phone;

**this**.hireDate = hireDate;

**this**.salary = salary;

**this**.email = email;

**this**.surname = surname;

**this**.department = department;

**this**.jobId = jobId;

**this**.managerId = managerId;

}

**public** **long** getId() {

**return** id;

}

**public** void setId(**long** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** void setName(String name) {

**this**.name = name;

}

**public** String getPhone() {

**return** phone;

}

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

**public** Date getHireDate() {

**return** hireDate;

}

**public** **void** setHireDate (Date hireDate) {

**this**.hireDate = hireDate;

}

**public** **long** getSalary() {

**return** salary;

}

**public** **void** setSalary(**long** salary) {

**this**.salary = salary;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getSurname() {

**return** surname;

}

**public** **void** setSurname(String surname) {

**this**.surname = surname;

}

**public** Department getDepartment() {

**return** department;

}

**public** **void** setDepartment(Department department) {

**this**.department = department;

}

**public** String getJobId () {

**return** jobId;

}

**public** **void** setJobId (String jobId) {

**this**.jobId = jobId;

}

**public** Manager getManagerId() {

**return** managerId;

}

**public** void setManagerId(Manager managerId) {

**this**.managerId = managerId;

}

@Override

**public** String toString() {

**return** String.*format*("id=%d, nombre=%s, apellido=%s, salario=%d, manager=%d, departamento=%d ", **this**.id, **this**.name, **this**.surname, **this**.salary, **this**.

managerId.getManager\_id(), **this**.department.getId());

}

@Override

**public** **int** hashCode() {

**return** (**int**)**this**.id;

}

}

**package** com.sopra.tutorialchapter3.entity;

**import** java.io.Serializable;

**import** java.util.List;

**import** javax.persistence.Access;

**import** javax.persistence.AccessType;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.OneToMany;

**import** javax.persistence.Table;

**import** javax.persistence.Transient;

**import** javax.persistence.Embedded;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.SequenceGenerator;

**import** com.sopra.tutorialchapter3.embededobject.Manager;

@Entity

@Table(name="Departments")

@Access(AccessType.***FIELD***)

**public** **class** Department **implements** Serializable{

@SequenceGenerator(name="Dep\_Gen", sequenceName="Dep\_Seq")

@Id @GeneratedValue(generator="Dep\_Gen")

**private** **long** department\_id;

@Transient

**private** String department\_name;

@OneToMany(mappedBy="department")

**private** List<Employee> employee;

@Column(name= "MANAGER\_ID")

@Embedded

**private** Manager managerId;

**public** Department() {

}

**public** Department(**long** department\_id, String department\_name, List<Employee>

employee, Manager managerId) {

**this**.department\_id = department\_id;

**this**.department\_name = department\_name;

**this**.employee = employee;

**this**.managerId = managerId;

}

**public** **long** getId() {

**return** department\_id;

}

**public** **void** setId(**long** department\_id) {

**this**.department\_id = department\_id;

}

**public** String getName() {

**return** department\_name;

}

**public** **void** setName(String department\_name) {

**this**. department\_name = department\_name;

}

@Access(AccessType.***PROPERTY***) @Column(name="DEPARTMENT\_NAME")

**public** String getNameWithDep (){

**if**(department\_name.length()<10) **return** "dep. " + department\_name;

**else** **return** department\_name;

}

**public** **void** setNameWithDep(String name){

**if**(name.contains("dep. ")) **this**.department\_name= name.substring(5);

**else** **this**.department\_name=name;

}

**public** List<Employee> getEmployee() {

**return** employee;

}

**public** **void** setEmployee(List<Employee> employee) {

**this**.employee = employee;

}

**public** Manager getManagerId() {

**return** managerId;

}

**public** **void** setManagerId(Manager managerId){

**this**.managerId = managerId;

}

@Override

**public** String toString() {

**return** String.*format*("id=%d, nombre=%s”,

**this**. department\_id, **this**.department\_name);

}

@Override

**public** **int** hashCode() {

**return** (**int**)**this**.department\_id;

}

}

Por último, la clase del objeto embebido *Manager* sería así:

**package** com.sopra.tutorialchapter3.embebedobject;

**import** javax.persistence.Access;

**import** javax.persistence.AccessType;

**import** javax.persistence.Embeddable;

**import** javax.persistence.Column;

@Embeddable @Access(AccessType.***PROPERTY***)

**public class** Manager {

**private int** id;

**public** Manager() {

}

**public** Manager(**int** id) {

**this**.id = id;

}

**public** **int** getManager\_id() {

**return** id;

}

**public** void setManager\_id(**int** id) {

**this**.id = id;

}

}

Se observa que gracias al acceso por propiedad (método *getter*), se puede accede a la columna adecuada sin hace ruso de la anotación *@Column*.

## Método main() para comprobar funcionamiento

Por último, con el objetivo de comprobar que la base de datos responde a todo lo creado en Java, se creará en un paquete nuevo la clase *Program* que contendrá un método *main()* de prueba en el cual se creará un empleado (y luego se comprobará en la base de datos que el identificador se genera correctamente). Este también hará la misma comprobación con *Department*.

Además, se comprobará que se puede obtener información de un empleado existente en la base de datos a partir de la sobreescritura del método *toString* que tienen las entindades. Este método se ha diseñado para que devuelva el id, el nombre, el apellido, el salario, el id del manager al que pertenece (el objeto embebido) y el id del departmaneto al que pertenece (la entidad relacionada).

Tener en cuenta que en "xx" irá el nombre de la unidad de persistencia que se haya escogido en *persistance.xml*.

**package** com.sopra.tutorialchapter3.program;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** javax.persistence.EntityManager;

**import** javax.persistence.EntityManagerFactory;

**import** javax.persistence.Persistence;

**import** com.sopra.tutorialchapter3.entity.Department;

**import** com.sopra.tutorialchapter3.entity.Employee;

**import** com.sopra.tutorialchapter3.embededobject.Manager;

**public** **class** Program {

**public** **static** **void** main(String[] args) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("xx");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

//Crear objeto Manager

Manager m = new Manager();

m.setManager\_id(100);

//Crear empleado

Employee aa= **new** Employee();

aa.setName("Javier");

aa.setSurname("Martinez");

aa.setEmail("jav9@soprasteria.com");

aa.setJobId("AD\_PRES");

aa.setSalary(2000L);

String pattern = "yyyy-MM-dd";

SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern);

Date hireDate=**null**;

**try** {

hireDate = simpleDateFormat.parse("2018-09-09");

} **catch** (ParseException e) {

e.printStackTrace();

}

aa.setHireDate(hireDate);

aa.setManagerId(m);

em.persist(aa);

//Crear departmaneto

Department d = **new** Department();

d.setName("IT");

d.setManagerId(m);

em.persist(d);

//Encontrar empleado

Employee found=em.find(Employee.**class**, 104L);

System.***out***.println(found);

em.getTransaction().commit();

em.close();

emf.close();

}

}