

ORM

Correspondencia

Objeto-relacional

Tema 3. Acceso a datos

Objetivos

- Comprender el propósito de los ORM
- Utilizar Hibernate para la persistencia de objetos en un esquema relacional.
- Crear POJOs apropiados para Hibernate
- Analizar el ciclo de vida de un objeto persistente en Hibernate
- Conocer los fundamentos de los lenguajes HQL y JPQL

Correspondencia Objeto Relacional

- Técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia

Java Persistence API

- JPA, es la API de persistencia desarrollada para la plataforma Java EE.
- El objetivo es no perder las ventajas de la orientación a objetos al interactuar con una base de datos (siguiendo el patrón de mapeo objeto-relacional) y permitir usar objetos regulares.
- Hibernate es una implementación de JPA

Hibernate

- Es un framework ORM para Java con licencia LGPL



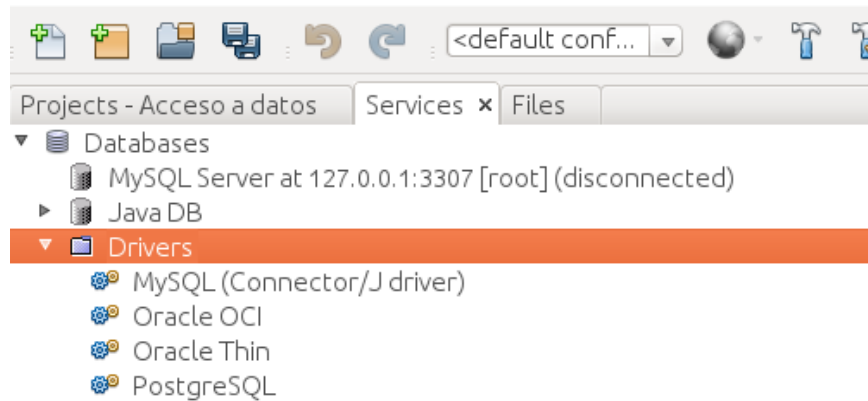
HIBERNATE

<https://hibernate.org/>

Objetos principales en Hibernate

- Session, SessionFactory
- Transaction
- Query

Conexión con base de datos en NetBeans



Customize Connection

Driver Name:

Host: Port:

Database:

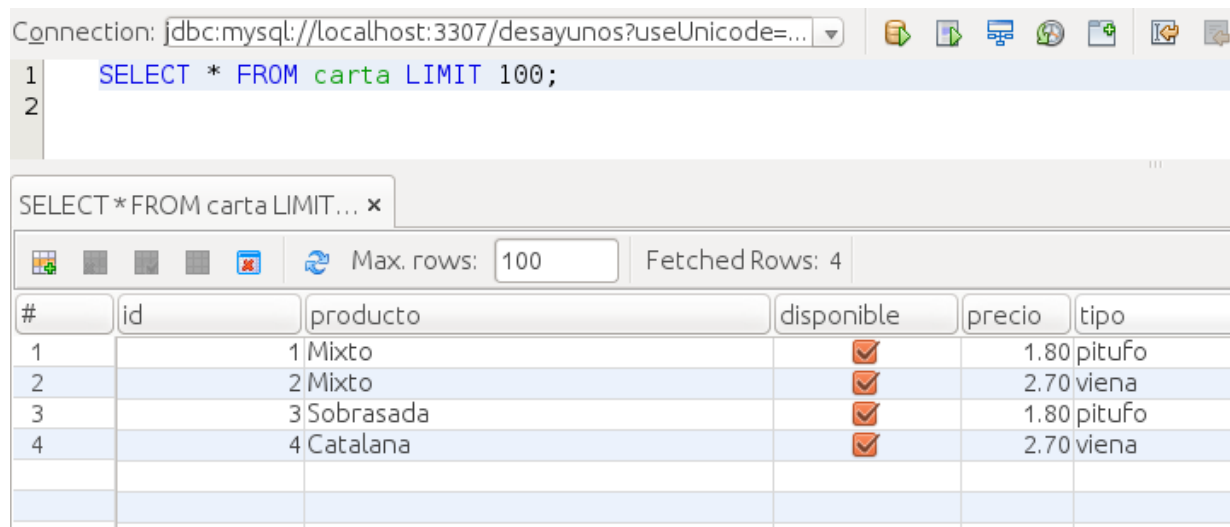
User Name:

Password:

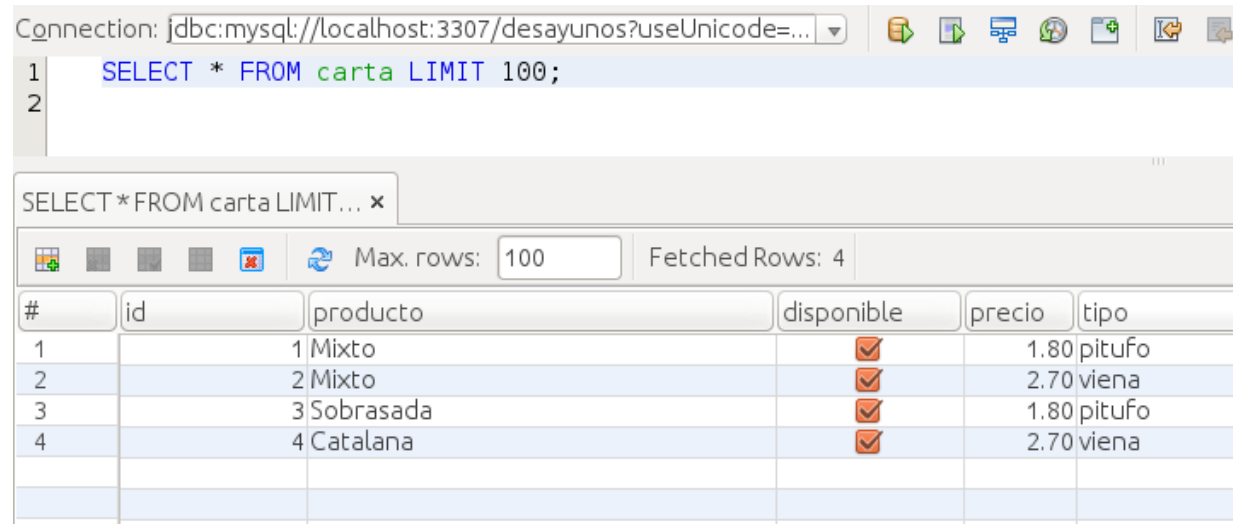
☐ Remember password

JDBC URL:

Conexión con base de datos en NetBeans



Conexión con base de datos en NetBeans



Nuevo proyecto

The screenshot displays the NetBeans 8.2 Plugin Portal interface. The 'Available Plugins (42)' tab is active. A table lists various plugins, with 'Hibernate' selected. The right panel shows details for the 'Hibernate' plugin, including its version (1.33.1.1), date (30/9/16), source (NetBeans 8.2 Plugin Portal), and homepage (<http://www.netbeans.org/>). The 'Install' button is highlighted, and a message indicates '1 plugin selected, 3MB'.

Updates (1) Available Plugins (42) Downloaded Installed (12) Settings

Check for Newest

Search:

| Install | Name | Category | Sou... |
|-------------------------------------|-----------------------------|------------------|--------|
| <input type="checkbox"/> | Perforce | io.github.tri... | |
| <input type="checkbox"/> | The nb-javac Java editin... | Java | |
| <input type="checkbox"/> | sonarlint4netbeans | Java | |
| <input type="checkbox"/> | Java Card™ | Java Card™ | |
| <input type="checkbox"/> | Oberthur Java Card Plat... | Java Card™ | |
| <input type="checkbox"/> | Gluon Plugin | JavaFX 2 | |
| <input type="checkbox"/> | Java ME Keystore Mana... | Java ME | |
| <input type="checkbox"/> | Java ME Common Ant T... | Java ME | |
| <input type="checkbox"/> | Java ME Core | Java ME | |
| <input type="checkbox"/> | Java ME Project | Java ME | |
| <input checked="" type="checkbox"/> | Hibernate | Java SE | |
| <input type="checkbox"/> | Linux notifications | Notifications | |
| <input type="checkbox"/> | PHP Enhancements | PHP | |
| <input type="checkbox"/> | CakePHP3 Framework | PHP | |
| <input type="checkbox"/> | PHP CS Fixer | PHP | |
| <input type="checkbox"/> | PHP WordPress Blog/CMS | PHP | |
| <input type="checkbox"/> | PHP CakePHP Framework | PHP | |

Hibernate

Certified Plugin

Version: 1.33.1.1
Date: 30/9/16
Source: NetBeans 8.2 Plugin Portal
Homepage: <http://www.netbeans.org/>

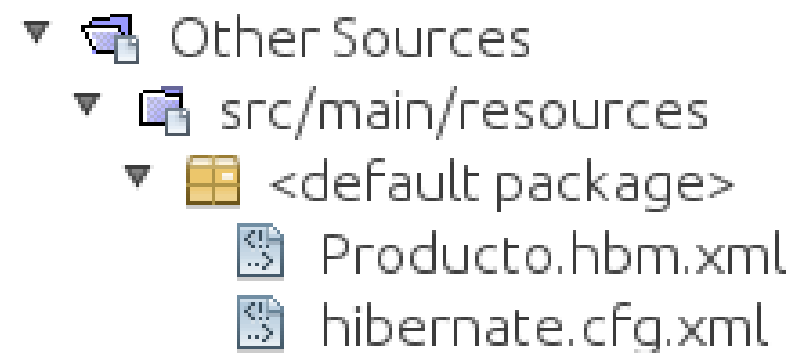
Plugin Description

This module provides Hibernate support in NetBeans

Install 1 plugin selected, 3MB

Archivos de configuración de hibernate

- Archivo de configuración
- Archivos de mapeo de clases
- Archivo POM (solo en el caso de usar Maven)



Archivos de configuración de hibernate

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
  <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
  <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
  <property name="hibernate.connection.url">jdbc:mysql://localhost:3307/desayunos?
    useUnicode=true&amp;useJDBCCompliantTimezoneShift=true&amp;
    useLegacyDatetimeCode=false&amp;serverTimezone=UTC</property>
  <property name="hibernate.connection.username">root</property>
  <property name="hibernate.connection.password">pacoromero</property>

  <mapping resource="Producto.hbm.xml"/>

</session-factory>
</hibernate-configuration>
```

Archivos de mapeo de hibernate

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="models.Producto" table="carta" >
    <id column="id" name="id" type="integer"/>
    <property column="producto" name="producto" type="string" />
    <property column="tipo" name="tipo" type="string" />
    <property column="disponible" name="disponible" type="boolean"/>
    <property column="precio" name="precio" type="big_decimal"/>
  </class>
</hibernate-mapping>
```

Mapecto mediante anotaciones

```
@Entity
@Table(name = "people")
public class Persona implements Serializable{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private long id;

    @Column(name = "nombre")
    private String nombre;

    @Column(name = "apellido")
    private String apellido;

    @Column(name = "edad")
    private Integer edad;
```

Maven Artifacts

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.5.2</version>
</dependency>

<dependency>
  <groupId>javassist</groupId>
  <artifactId>javassist</artifactId>
  <version>3.4.GA</version>
  <scope>compile</scope>
</dependency>

<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-annotations</artifactId>
  <version>3.4.0.GA</version>
  <scope>compile</scope>
</dependency>
```

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.4.22.Final</version>
</dependency>

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.22</version>
</dependency>
```



Maven Artifacts

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>2.3.2</version>
      <configuration>
        <showDeprecation>>true</showDeprecation>
      </configuration>
    </plugin>
  </plugins>
</build>
```


Classes POJO

```
package models;

import ...5 lines

public class Producto implements Serializable {

    private Integer id;
    private String producto;
    private String tipo;
    private Boolean disponible;
    private BigDecimal precio;

    public Producto() {...7 lines }
    public Producto(Integer id, String producto, String tipo, Boolean disponible, double precio)
    public Integer getId() {...3 lines }
    public void setId(Integer id) {...3 lines }
    public String getProducto() {...3 lines }
    public void setProducto(String producto) {...3 lines }
    public String getTipo() {...3 lines }
    public void setTipo(String tipo) {...3 lines }
    public Boolean getDisponible() {...3 lines }
    public void setDisponible(Boolean disponible) {...3 lines }
    public BigDecimal getPrecio() {...3 lines }
    public void setPrecio(BigDecimal precio) {...3 lines }
    public void setPrecio(double precio) {...3 lines }
```

HibernateUtil.java

```
import org.hibernate.HibernateException;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {

    private static final SessionFactory sessionFactory;

    static {
        try {
            sessionFactory = new Configuration().configure("hibernate.cfg.xml").buildSessionFactory();
        } catch (HibernateException ex) {
            // Log the exception.
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Flujo de uso de Hibernate

- Inicialización de logs
- Obtener la sesión
- Iniciar una transacción
- Operar con los datos

Clase Session

.beginTransaction()
 .save()
 .delete()
 .createQuery()
 .close()

Clase Transaction

`.commit()`

`.rollback()`

Flujo de uso de Hibernate

```
org.apache.log4j.BasicConfigurator.configure();

try (Session s = HibernateUtil.getSessionFactory().openSession()) {
    Transaction t = null;

    try {
        t = s.beginTransaction();

        models.Producto bocata = new models.Producto();
        bocata.setTipo("Viena");
        bocata.setPrecio(2);
        bocata.setProducto("Zurrapa");
        bocata.setDisponible(true);

        s.save(bocata);

        t.commit();
    }
}
```

Mapeo de las relaciones

- <one-to-one> Indica que el objeto es parte de una relación simple
- <many-to-one> Indica que el objeto es parte de una relación N-1.
- <one-to-many> Indica que el objeto es parte de una relación 1-N. En este caso el atributo sería el lado N
- <many-to-many> Indica que el objeto es parte de una relación N-M. En este caso se indica la tabla que mantiene la referencia entre las tablas y los campos que hacen el papel de claves ajenas en la base de datos

Claves primarias

```
<id name="id" type="Integer">  
  <column name="id_columna" />  
  <generator class="tipo_generador">  
    <param name="property">valor</param>  
  </generator>  
</id>
```


Clases de generadores

- **Identity:** se usa conjunto al auto_increment de mysql
- **Assigned:** cuando la aplicación debe proporcionar un valor para el campo
- **Increment:** se calcula el siguiente id mediante una consulta sql
- **Foreign:** para las relaciones 1 a 1
- **Secuence:** usa secuencia de la base de datos al estilo Postgre u Oracle.

Sesiones y estados los objetos persistentes

Transitorio: el objeto se acaba de crear y no esta asociado a ningun contexto de persistencia. Podría tener un id si el generador es de tipo assigned.

Persistente: el objeto tiene un identificador y esta asociado a un contexto de persistencia. Esta almacenado en la base de datos. Cualquier cambio que se realice sobre el, se almacenara en la base de datos.

Sesiones y estados los objetos persistentes

Separado: el objeto tiene identificador, pero no dispone de un contexto de persistencia. Los cambios realizados sobre el no se actualizan en la base de datos.

Eliminado: el objeto tiene un identificador y esta asociado a un contexto de persistencia, pero esta pendiente de ser eliminado.

Sesiones y estados los objetos persistentes

De transitorio a persistente:

```
int session.save(obj);
```

```
int session.saveOrUpdate(obj);
```

```
session.persist(obj);
```

Sesiones y estados los objetos persistentes

Obtención de un objeto persistente

`session.get(class,id);` (Si no existe --> null)

`session.load(class,id);` (Si no existe --> `ObjectNotFoundException`)

De persistente a eliminado

`session.delete(class,id);`

Sesiones y estados los objetos persistentes

De persistente a separado

`session.evict(obj);` // separa

`session.close();` // graba y separa todo

`session.clear();` // separa todo sin grabar

Sesiones y estados los objetos persistentes

De separado a persistente

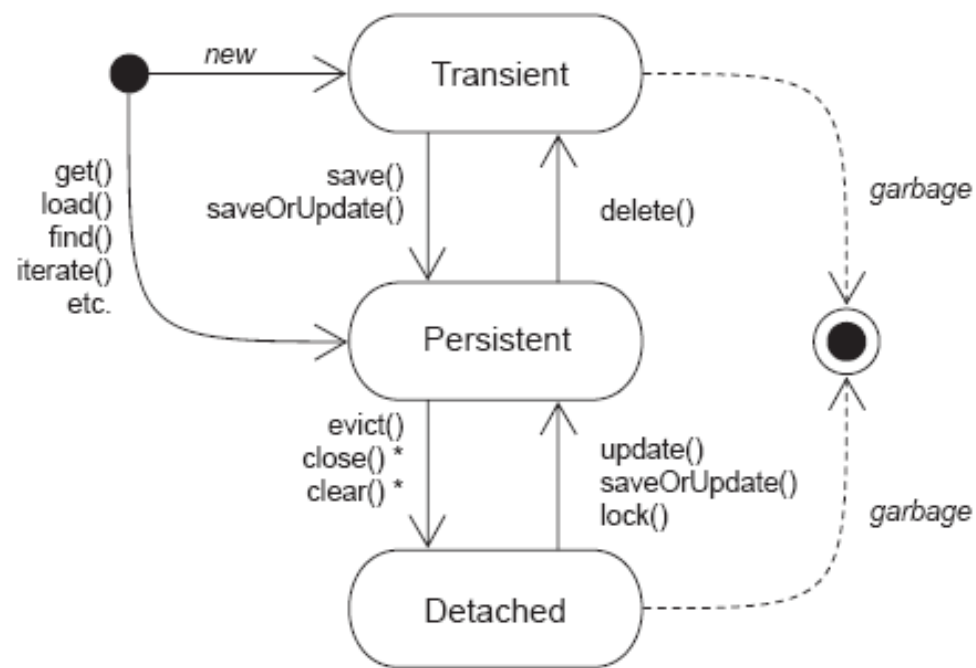
`session.update(obj);`

`session.lock(obj,modo);`

`session.saveOrUpdate(obj);`

`session.merge(obj);`

Sesiones y estados los objetos persistentes



* affects all instances in a Session

Ejemplo one-to-many

| desayunos carta |
|---------------------------|
| id : int |
| producto : text |
| # disponible : tinyint(1) |
| # precio : decimal(5,2) |
| tipo : varchar(64) |

| desayunos pedidos |
|----------------------|
| id : int |
| alumno : text |
| ciclo : varchar(16) |
| 1 fecha : date |
| # producto_id : int |
| estado : varchar(32) |

one → many

Ejemplo one-to-many

La implementación de esta relación es doble:

- En la clase pedido, mediante el atributo **id_producto**, que establece a que fila de la tabla producto le corresponde.
- En la clase producto, mediante el atributo **pedidos** de tipo Set que indica el conjunto de pedidos en los que esta presente dicho producto.

Ejemplo one-to-many

```
<hibernate-mapping>
  <class name="models.Producto" table="carta" >
    ...
    <set name="pedidos" table="pedido" inverse="true">
      <key>
        <column name="producto_id"></column>
      </key>
      <one-to-many class="models.Pedido"></one-to-many>
    </set>
  </class>
</hibernate-mapping>
```

Ejemplo one-to-many

```
public class Producto implements Serializable {  
  
    private Integer id;  
    private String producto;  
    private String tipo;  
    private Boolean disponible;  
    private BigDecimal precio;  
  
    private Set pedidos;  
  
    public Producto(){  
        this.pedidos = new HashSet(0);  
        this.id=0;  
    }  
}
```

Ejemplo one-to-many

```
<hibernate-mapping>
  <class name="models.Pedido" table="pedidos" >
    <id column="id" name="id" type="integer"/>
    <property column="alumno" name="alumno" type="string" />
    <property column="ciclo" name="ciclo" type="string" />
    <property column="fecha" name="fecha" type="date"/>
    <property column="estado" name="estado" type="string"/>

    <many-to-one name="producto" class="models.Producto" fetch="select">
      <column name="producto_id"/>
    </many-to-one>

  </class>
</hibernate-mapping>
```

Ejemplo one-to-many con anotaciones

| university |
|------------|
| id |
| name |

| student |
|---------------|
| id |
| name |
| university_id |

Ejemplo many-to-one

```
1 @Entity
2 @Table(name = "students")
3 public class Student {
4
5     @Id
6     @GeneratedValue(strategy = GenerationType.IDENTITY)
7     private Long id;
8
9     private String name;
10
11     @ManyToOne()
12     @JoinColumn(name = "university_id")
13     private University university;
14
15     /* Getters and setters */
16 }
```

Ejemplo one-to-many

```
1 @Entity
2 @Table(name = "university")
3 public class University {
4
5     @Id
6     @GeneratedValue(strategy = GenerationType.IDENTITY)
7     private Long id;
8
9     private String name;
10
11     @OneToMany(mappedBy = "university", cascade = CascadeType.ALL, orphanRemoval = true)
12     private List<Student> students;
13
14     /* Getters and setters */
```


Uso de iteradores

```
Iterator<models.Pedido> pedidos= p1.getPedidos().iterator();  
while(pedidos.hasNext()){  
    System.out.println(pedidos.next().toString());  
}
```

Ejemplo many-to-many



Ejemplo many-to-many

La implementación de esta relación es doble y necesita una tabla auxiliar.

La relación muchos a muchos consiste en que un objeto A tenga una lista de otros objetos B y también que el objeto B a su vez tenga la lista de objetos A.

Al persistirse cualquier objeto (A o B) también se persiste la lista de objetos que posee.

Ejemplo many-to-many

```
public class Evento implements Serializable {  
    private Integer id;  
    private String nombre;  
    private Date fecha;  
    private Set<models.Persona> personas;  
  
    public Evento() {  
        this.personas = new HashSet();  
    }  
  
    public Evento(Integer id, String nombre, Date fecha) {  
        this.personas = new HashSet();  
        this.id = id;  
        this.nombre = nombre;  
        this.fecha = fecha;  
    }  
}
```

Ejemplo many-to-many

```
public class Persona implements Serializable{  
    private Integer id;  
    private String nombre;  
    private String apellidos;  
    private Set<models.Evento> eventos;  
  
    public Persona(Integer id, String nombre, String apellidos) {  
        this.eventos = new HashSet();  
        this.id = id;  
        this.nombre = nombre;  
        this.apellidos = apellidos;  
    }  
}
```

Ejemplo many-to-many

```
<class name="models.Persona" table="persona" >
  <id column="id" name="id" type="integer">
    <generator class="increment" />
  </id>
  <property name="nombre" type="string" />
  <property name="apellidos" type="string" />
  <set name="eventos" table="eventopersona" cascade="all" inverse="true" >
    <key>
      <column name="id_persona" />
    </key>
    <many-to-many column="id_evento" class="models.Evento" />
  </set>
</class>
```

Ejemplo many-to-many

```
<class name="models.Evento" table="evento" >
  <id column="id" name="id" type="integer">
    <generator class="increment" />
  </id>
  <property name="nombre" type="string" />
  <property name="fecha" type="date" />
  <set name="personas" table="eventopersona" cascade="all" inverse="false" >
    <key>
      <column name="id_evento" />
    </key>
    <many-to-many column="id_persona" class="models.Persona" />
  </set>
</class>
```

Ejemplo one-to-one

Para reflejar esta relación se usan atributos sencillos en ambas clases.

Ambas tablas comparten clave primaria

Se utiliza al campo one-to-one para referencias a la clase asociada.

Ejemplo one-to-one

```
public class Correo implements Serializable{  
    private Integer id;  
    private String email;  
    private Persona persona;  
  
    public Correo() {  
    }  
  
    public Correo(Integer id, String email) {  
        this.id = id;  
        this.email = email;  
    }  
}
```

Ejemplo one-to-one

```
<class name="models.Correo" table="email" >
  <id column="id" type="integer">
    <generator class="foreign">
      <param name="property">persona</param>
    </generator>
  </id>
  <property name="email" type="string" />
  <one-to-one name="persona" class="models.Persona" constrained="true"/>
</class>
```

Ejemplo one-to-one

```
Evento ev1 = new Evento();  
ev1.setFecha(new Date());  
ev1.setNombre("Navidad");  
s.save(ev1);
```

```
Persona p1 = new Persona();  
p1.setNombre("Francisco");  
p1.setApellidos("Romero");  
models.Correo co = new models.Correo();  
co.setEmail("paco@romero.com");  
p1.setCorreo(co);  
co.setPersona(p1);  
s.save(p1);
```

```
ev1.getPersonas().add(p1);
```



Consultas HQL

Hibernate Query Lenguaje

Es el lenguaje para realizar consultas en Hibernate y esta basado en SQL con el que comparte estructura y palabras clave.

Hay que tener en cuenta que no se trabaja directamente con la base de datos sino sobre los objetos y clases.

Se usa con el método **query** de la clase Query.



Interfaz Query

```
Query q = s.createQuery("FROM Persona WHERE nombre LIKE :nombre").  
    setParameter("nombre", "Julia").  
    setReadOnly(true).  
    setFirstResult(0).  
    setMaxResults(3);
```

```
List<models.Persona> julias = q.getResultList();
```

```
for(models.Persona p : julias){  
    System.out.println(p);  
}
```

Interfaz Query

List<T> getResultList()
T getSingleResult()
Int executeUpdate()
Query<T> setParameter(...)

Query<T> setReadOnly()
Query<T> setFirstResult()
Query<T> setMaxResult()

Ejemplos HQL

```
Query query = session.createQuery("from Stock where stockCode = :code ");  
query.setParameter("code", "7277");  
List<stock> list = query.list();
```

```
Query query = session.createQuery("update Stock set stockName = :stockName"  
+ " where stockCode = :stockCode");  
query.setParameter("stockName", "DIALOG1");  
query.setParameter("stockCode", "7277");  
int result = query.executeUpdate();
```

Ejemplos HQL

```
Query query = session.createQuery("delete Stock where stockCode = :stockCode");  
query.setParameter("stockCode", "7277");  
int result = query.executeUpdate();
```

```
Query q=session.createQuery("select sum(salary) from Emp");  
List<Integer> list=q.list();  
System.out.println(list.get(0));
```


Ejemplos HQL

```
Query query = session.createQuery("SELECT E.firstName FROM Employee E");  
List results = query.list();
```

```
Query q=session.createQuery("FROM Employee E WHERE E.id > 10 " +  
    "ORDER BY E.firstName DESC, E.salary DESC ");  
List list=q.list();
```

```
Query q=session.createQuery("SELECT count(distinct E.firstName) "+  
    "FROM Employee E ");  
List list=q.list();
```

Herencia

La herencia como tal no se puede representar directamente en un esquema relacional, hay que mapearla.

Tampoco se puede representar con un esquema E-R ni corresponder directamente con Hibernate.

Hay que transformar la relación usando eliminaciones:

- **Eliminación de subtipos**
- **Eliminación de jerarquía**

Herencia

```
public class Publicacion implements java.io.Serializable {  
    private Integer idPub;  
    private String nomPub;  
}  
public class Libro extends Publicacion implements java.io.Serializable {  
    private String isbn;  
    private String autor;  
}  
public class Revista extends Publicacion implements java.io.Serializable {  
    private String issn;  
}
```

Eliminación de subtipos

- Los atributos de los subtipos pasan al supertipo y deben poder ser null.
- Las relaciones con los subtipos pasan a ser con el supertipo.
- Es necesario incluir un nuevo atributo para discriminar el subtipo.

Eliminación de subtipos

```
create table publicacion(  
    id_pub integer auto_increment not null,  
    nom_pub varchar(50) not null,  
    tipo char(3) not null,  
    isbn char(13),  
    autor varchar(40),  
    issn char(10),  
    primary key(id_pub),  
    constraint check_subtipos check(tipo='pub' or (tipo='lib' and not isnull(isbn)  
                                   and not isnull(autor)) or (tipo='rev' and not isnull(issn)))  
);
```

Eliminación de subtipos

```
▼<hibernate-mapping>
  ▼<class name="ORM.Publicacion" table="publicacion" catalog="proyecto_orm" discriminator-value="pub">
    ▼<id name="idPub" type="java.lang.Integer">
      <column name="id_pub"/>
      <generator class="identity"/>
    </id>
    <discriminator column="tipo" type="string"/>
    ▼<property name="nomPub" type="string">
      <column name="nom_pub" length="50"/>
    </property>
    ▼<subclass name="ORM.Libro" discriminator-value="lib">
      ▼<property name="isbn" type="string">
        <column name="isbn" length="13"/>
      </property>
      ▼<property name="autor" type="string">
        <column name="autor" length="40"/>
      </property>
    </subclass>
    ▼<subclass name="ORM.Revista" discriminator-value="rev">
      ▼<property name="issn" type="string">
        <column name="issn" length="10"/>
      </property>
    </subclass>
  </class>
</hibernate-mapping>
```

Eliminación de subtipos

```
create table publicacion(  
  id_pub integer auto_increment not null,  
  nom_pub varchar(50) not null,  
  primary key(id_pub)  
);
```

```
create table libro(  
  id_pub integer not null,  
  isbn char(13) not null,  
  autor varchar(40),  
  primary key(id_pub),  
  constraint fk_libro_publicacion foreign key(id_pub) references publicacion(id_pub)  
);
```

```
create table revista(  
  id_pub integer not null,  
  issn char(10) not null,  
  primary key(id_pub),  
  constraint fk_revista_publicacion foreign key(id_pub)  
    references publicacion(id_pub)  
);
```

Eliminación de subtipos

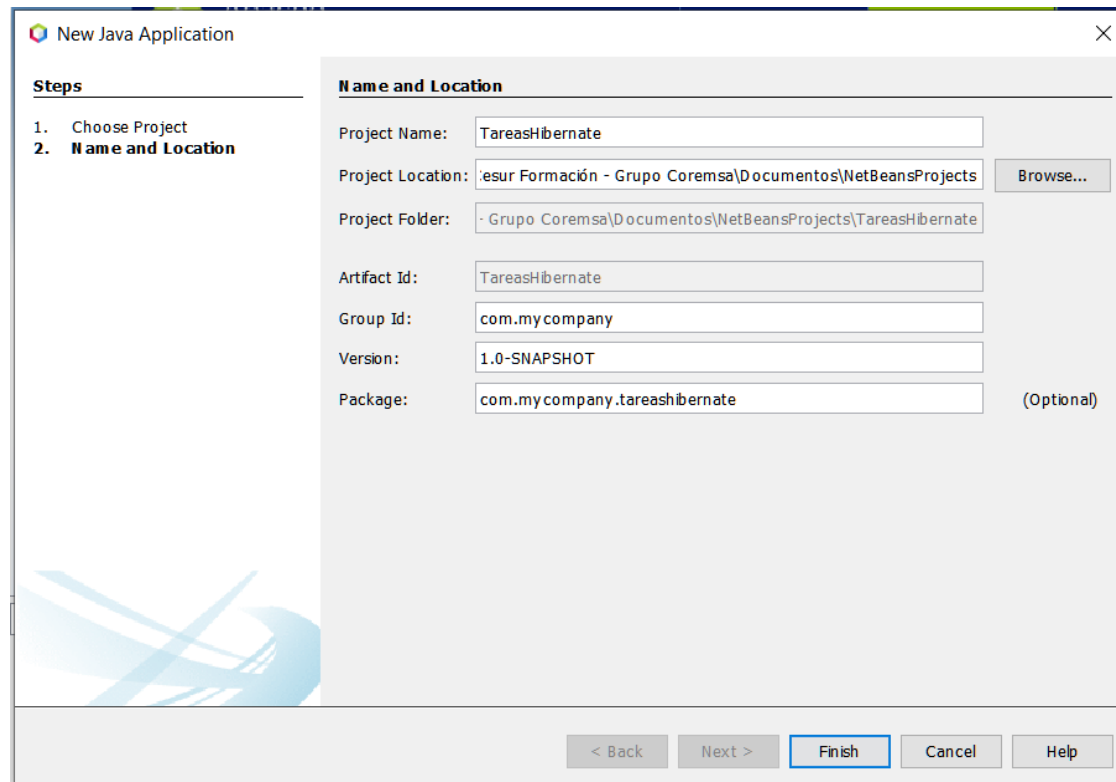
```
▼<hibernate-mapping>
  ▼<class name="ORM.Publicacion" table="publicacion" catalog="proyecto_orm">
    ▼<id name="idPub" type="java.lang.Integer">
      <column name="id_pub"/>
      <generator class="identity"/>
    </id>
    ▼<property name="nomPub" type="string">
      <column name="nom_pub" length="50"/>
    </property>
    ▼<joined-subclass name="ORM.Libro">
      <key column="id_pub"/>
      ▼<property name="isbn" type="string">
        <column name="isbn" length="13"/>
      </property>
      ▼<property name="autor" type="string">
        <column name="autor" length="40"/>
      </property>
    </joined-subclass>
    ▼<joined-subclass name="ORM.Revista">
      <key column="id_pub"/>
      ▼<property name="issn" type="string">
        <column name="issn" length="10"/>
      </property>
    </joined-subclass>
  </class>
</hibernate-mapping>
```


Ejemplo Paso a paso

| v | ad | tareas |
|---|----|-------------------------|
| # | | id : int(11) |
| | | nombre : varchar(30) |
| # | | alumno_id : int(11) |
| | | prioridad : varchar(16) |
| | | fecha : date |

| v | ad | alumno |
|---|----|----------------------|
| | | id : int(11) |
| | | nombre : varchar(30) |

Paso a paso



New Java Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: Tareashibernate

Project Location: esur Formación - Grupo Coremsa\Documentos\NetBeansProjects Browse...

Project Folder: - Grupo Coremsa\Documentos\NetBeansProjects\Tareashibernate

Artifact Id: Tareashibernate

Group Id: com.mycompany

Version: 1.0-SNAPSHOT

Package: com.mycompany.tareashibernate (Optional)

< Back Next > Finish Cancel Help

Paso a paso



Hibernate Core Relocation » 5.6.0.Final

Hibernate's core ORM functionality

| | |
|--------------|--|
| License | LGPL 2.1 |
| Categories | Object/Relational Mapping |
| Organization | Hibernate.org |
| HomePage | https://hibernate.org/orm |
| Date | (Oct 11, 2021) |
| Files | pom (5 KB) jar (7.1 MB) View All |
| Repositories | Central |
| Used By | 3,558 artifacts |

Paso a paso

[Home](#) » [org.hibernate](#) » [hibernate-annotations](#) » [3.5.6-Final](#)



Hibernate Annotations » 3.5.6-Final

Annotations metadata for Hibernate

| | |
|--------------|---|
| License | LGPL 2.1 |
| Date | (Sep 15, 2010) |
| Files | pom (7 KB) jar (356 KB) View All |
| Repositories | Central JBoss Public JBoss Releases |
| Used By | 670 artifacts |

[Maven](#)

[Gradle](#)

[Gradle \(Short\)](#)

[Gradle \(Kotlin\)](#)

[SBT](#)

[Ivy](#)

[Grape](#)

[Leiningen](#)

[Buildr](#)

```
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-annotations -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-annotations</artifactId>
  <version>3.5.6-Final</version>
</dependency>
```



Paso a paso

[Home](#) » [mysql](#) » [mysql-connector-java](#) » 8.0.26



MySQL Connector/J » 8.0.26

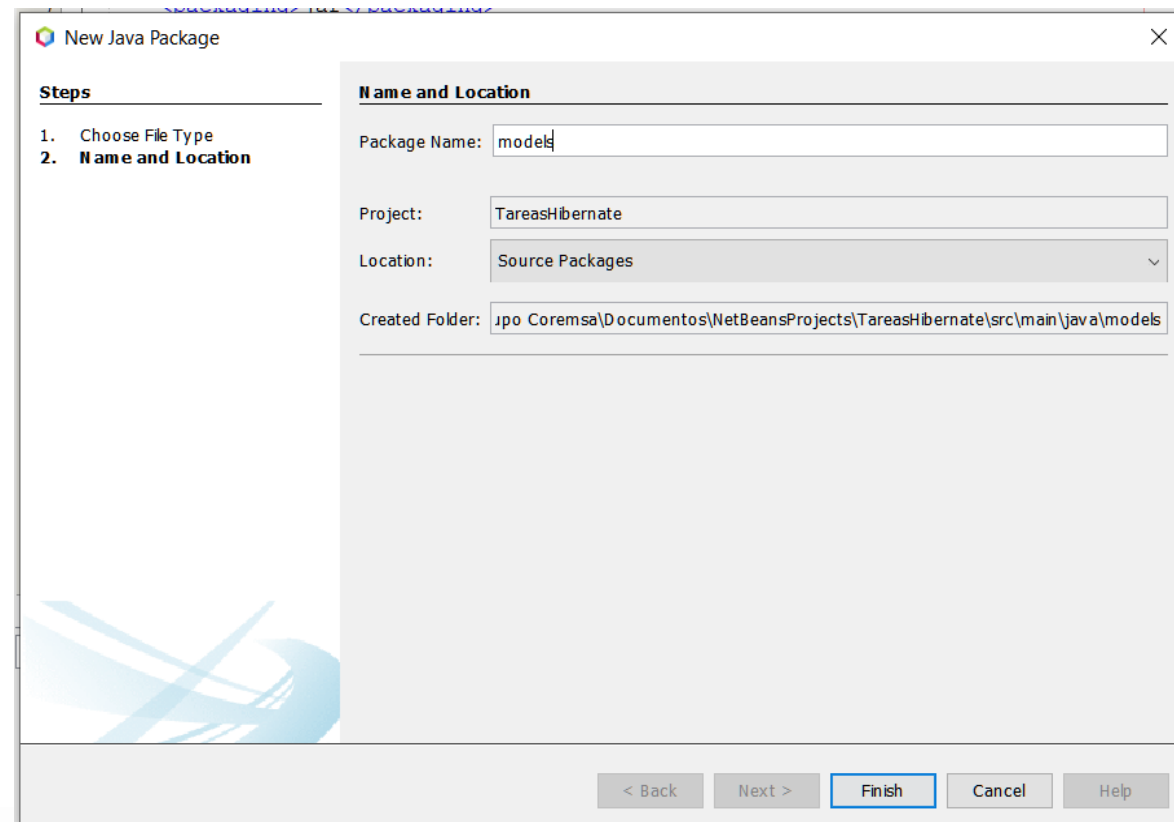
JDBC Type 4 driver for MySQL

| | |
|--------------|---|
| License | GPL 2.0 |
| Categories | MySQL Drivers |
| Organization | Oracle Corporation |
| HomePage | http://dev.mysql.com/doc/connector-j/en/ |
| Date | (Jul 19, 2021) |
| Files | pom (2 KB) jar (2.3 MB) View All |
| Repositories | Central |
| Used By | 5,646 artifacts |

Paso a paso

```
<dependencies>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.6.0.Final</version>
  </dependency>
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-annotations</artifactId>
    <version>3.5.6-Final</version>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.26</version>
  </dependency>
</dependencies>
```

Paso a paso



New Java Package

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Package Name:

Project:

Location:

Created Folder:

< Back Next > **Finish** Cancel Help

Paso a paso

New Java Class

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name:

Project:

Location:

Package:

Created File:

Superclass:

Interfaces:

Paso a paso

```
package models;  
  
import java.io.Serializable;  
  
public class Alumno implements Serializable {  
  
    private Integer id;  
    private String nombre;  
  
    public Alumno(Integer id, String nombre) {  
        this.id = id;  
        this.nombre = nombre;  
    }  
  
    public Alumno() {  
    }  
}
```

Paso a paso

```
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import static javax.persistence.GenerationType.IDENTITY;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="alumno")
public class Alumno implements Serializable {

    @Id
    @GeneratedValue(strategy = IDENTITY)
    private Integer id;

    @Column(name="nombre")
    private String nombre;
```

Paso a paso

```
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import static javax.persistence.GenerationType.IDENTITY;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="alumno")
public class Alumno implements Serializable {

    @Id
    @GeneratedValue(strategy = IDENTITY)
    private Integer id;

    @Column(name="nombre")
    private String nombre;
```

Paso a paso

New Java Class

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name:

Project:

Location:

Package:

Created File:

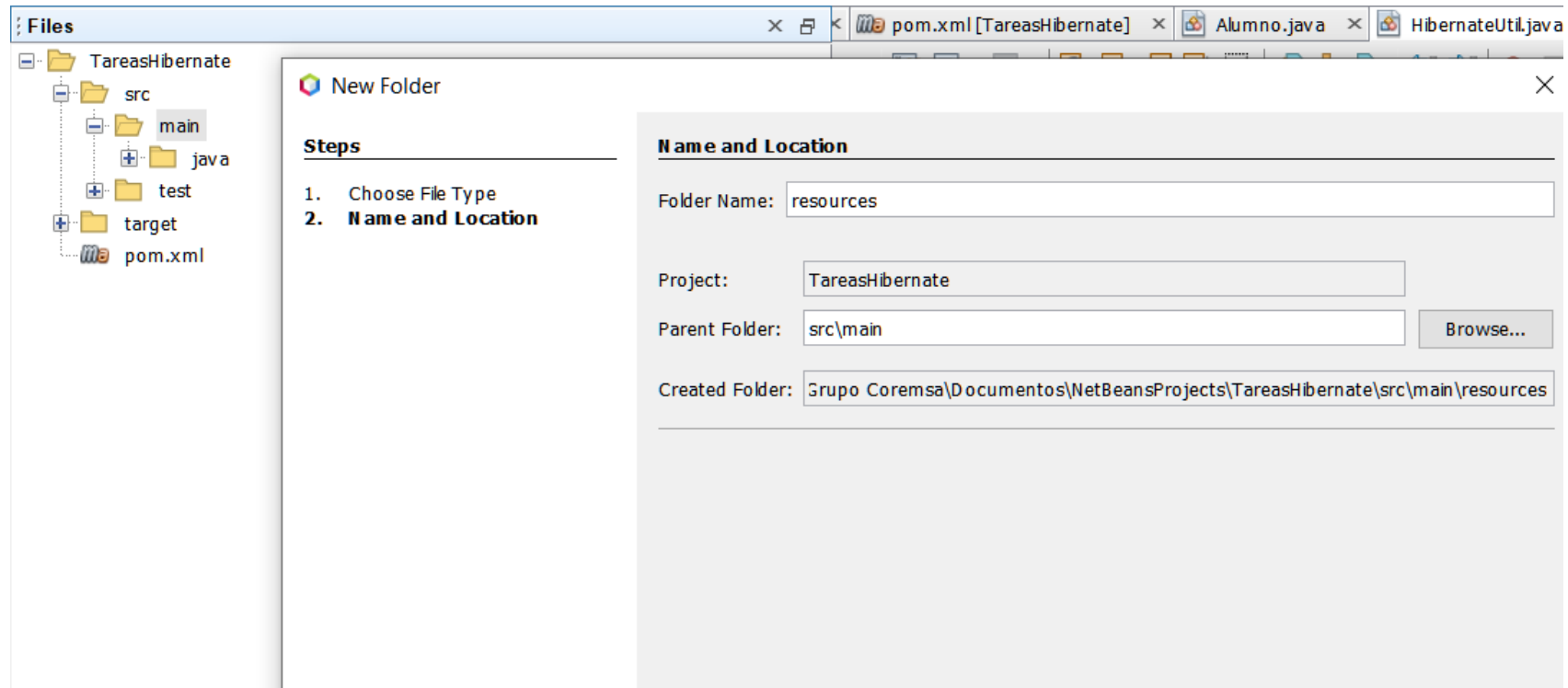
Superclass:

Interfaces:

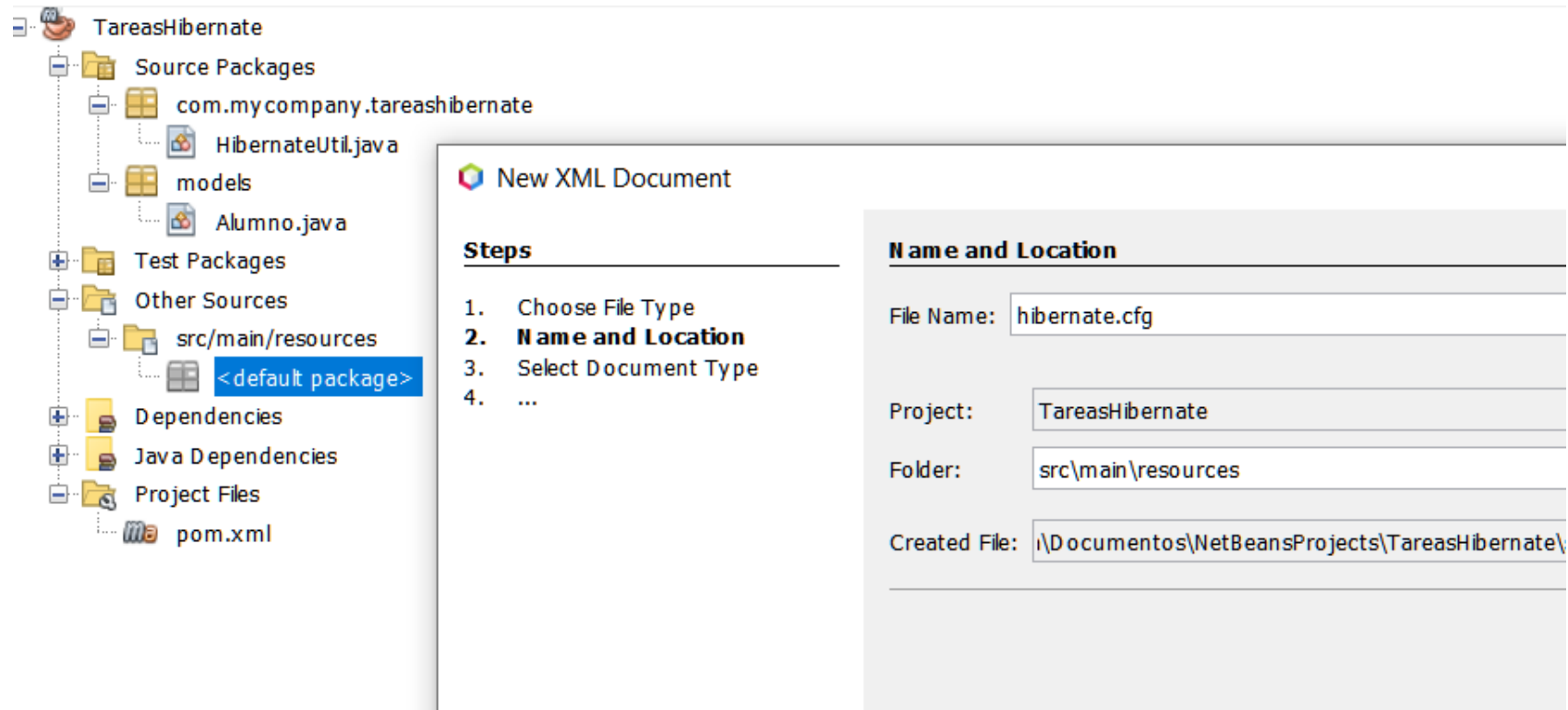
Paso a paso

```
public class HibernateUtil {  
  
    private static final SessionFactory sessionFactory;  
  
    static {  
        try {  
            sessionFactory = new Configuration().configure().buildSessionFactory();  
        } catch (HibernateException ex) {  
            System.err.println("Initial SessionFactory creation failed." + ex);  
            throw new ExceptionInInitializerError(ex);  
        }  
    }  
  
    public static SessionFactory getSessionFactory() {  
        return sessionFactory;  
    }  
  
}
```

Paso a paso



Paso a paso



The screenshot shows the NetBeans IDE interface. On the left, the 'TareasHibernate' project is expanded, showing a hierarchy of 'Source Packages' (com.mycompany.tareashibernate), 'Test Packages', 'Other Sources', 'Dependencies', 'Java Dependencies', and 'Project Files'. The 'src/main/resources' folder is selected, and a '<default package>' option is visible. On the right, the 'New XML Document' dialog is open, displaying the following information:

| Steps | |
|-------|--------------------------|
| 1. | Choose File Type |
| 2. | Name and Location |
| 3. | Select Document Type |
| 4. | ... |

| Name and Location | |
|-------------------|---|
| File Name: | hibernate.cfg |
| Project: | TareasHibernate |
| Folder: | src/main/resources |
| Created File: | I:\Documentos\NetBeansProjects\TareasHibernate\ |

Paso a paso

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>

    <property name = "hibernate.dialect">
      org.hibernate.dialect.MySQLDialect
    </property>

    <property name = "hibernate.connection.driver_class">
      com.mysql.cj.jdbc.Driver
    </property>

    <property name = "hibernate.connection.url">
      jdbc:mysql://localhost/ad
    </property>

    <property name = "hibernate.connection.username">
      root
    </property>

    <property name = "hibernate.connection.password">
    </property>

    <property name="show_sql">true</property>

    <mapping class = "models.Alumno"/>

  </session-factory>
</hibernate-configuration>
```


Paso a paso

TareasHibernate

- Source Packages
 - com.mycompany.tareashibernate
 - HibernateUtil.java
 - models
 - Alumno.java
- Test Packages
- Other Sources
 - src/main/resources
 - <default package>
- Dependencies
- Java Dependencies
- Project Files
 - pom.xml

New XML Document

Steps

1. Choose File Type
2. **Name and Location**
3. Select Document Type
4. ...

Name and Location

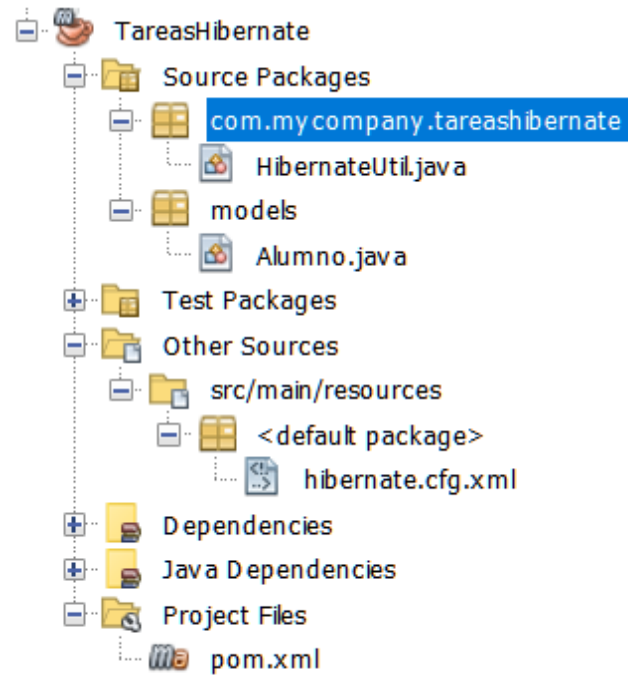
File Name:

Project:

Folder:

Created File:

Paso a paso



New Java Main Class

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

| | |
|---------------|---|
| Class Name: | Principa |
| Project: | TareasHibernate |
| Location: | Source Packages |
| Package: | com.mycompany.tareashibernate |
| Created File: | rojects\TareasHibernate\src\main\java\com\my comp |
| Superclass: | |
| Interfaces: | |

Paso a paso

```
public class Principal {  
  
    public static void main(String[] args) {  
  
        Session s = HibernateUtil.getSessionFactory().openSession();  
  
        Alumno a = new Alumno(0, "David");  
  
        Transaction t = s.beginTransaction();  
        s.save(a);  
        t.commit();  
    }  
}
```

Paso a paso

```
oct 18, 2021 10:14:10 P. M. org.hibernate.dialect.Dialect <init>  
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect  
oct 18, 2021 10:14:11 P. M. org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformIni  
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform  
Hibernate: insert into alumno (nombre) values (?)
```

```
-----  
BUILD SUCCESS  
-----
```

```
Total time: 5.528 s  
Finished at: 2021-10-18T22:14:11+02:00  
-----
```



Paso a paso

```
1 SELECT * FROM alumno LIMIT 100;
```

SELECT * FROM alumno LIM... X



Max. rows: 100

Fetches Rows: 1

| # | id | nombre |
|---|----|--------|
| 1 | 1 | David |
| | | |
| | | |
| | | |