

Acceso a Datos

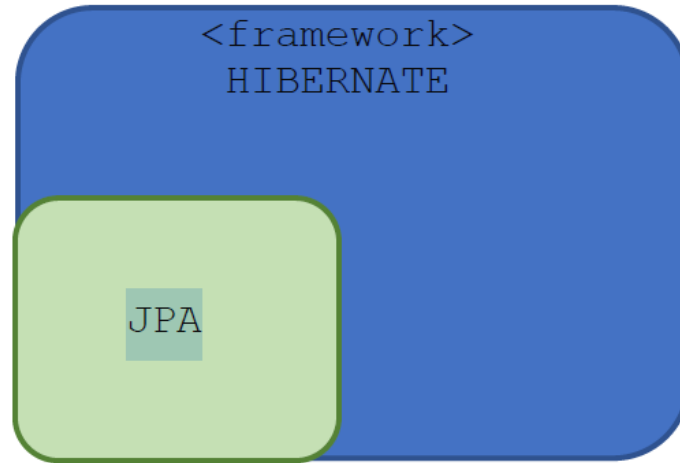
UT4. JPA

1. Introducción

- Muchas personas que se han introducido en el mundo de la persistencia se preguntan: ¿no son Hibernate y JPA lo mismo? Realmente, la diferencia es mucha, ya que JPA es una especificación que carece de implementación. Hibernate, por contra, es un producto muy real, un framework con una serie de funcionalidades muy concretas.
- Entonces, ¿qué relación tienen? Hibernate, a través de uno de sus módulos, proporciona una implementación de la especificación JPA, de forma que podemos usar JPA como interfaz, sabiendo que por debajo, las operaciones con la base de datos se estarán realizando con Hibernate. De alguna manera, Hibernate es de facto la implementación de referencia de JPA.

1. Introducción

- Es decir, Hibernate implementa como parte de su código la especificación de JPA. Podemos usar Hibernate para construir una capa de persistencia apoyándonos en las definiciones y reglas que la especificación de JPA, aunque no es obligatorio.



1. Introducción

La especificación JPA define lo siguiente:

- Una *facilidad* para especificar cómo nuestros objetos Java se relacionan con el esquema de una base de datos (a través de XML o de anotaciones).
- Una API sencilla para realizar las operaciones CRUD (create, read, update, delete), haciendo uso de un `javax.persistence.EntityManager`.
- Un lenguaje y una API para realizar consultas sobre los datos. El lenguaje, llamado JPQL (Java Persistence Query Language) se parece mucho a SQL.

Hibernate incluye, como parte de su código, toda la especificación JPA. Es decir, podemos usar Hibernate como motor de persistencia de JPA; sin embargo, incluye más funcionalidades que las que define la especificación.

2. Proyecto de JPA con Hibernate

Seguimos los mismos pasos que en la lección pasada, creando el proyecto Maven. En este caso, los datos pueden ser:

- GroupId: com.iesvjp.hibernate (o algo similar)
- ArtifactId: PrimerEjemploHibernateJpa (o algo similar)
- Package: com.iesvjp.hibernate.primerejemplohibernatejpa (se trata de la concatenación de groupId y artifactId, pero todo en minúscula).

Actualizamos la versión de Java, y pasamos a configurar el pom.xml.

2. Proyecto de JPA con Hibernate

En este caso, las dependencias que incluiremos son las siguientes:

```
<dependencies>
  <!-- Otras dependencias, como la de junit... -->
  <!-- for JPA, use hibernate-entitymanager instead of hibernate-core -->
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>5.2.9.Final</version>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.20</version>
  </dependency>
</dependencies>
```

2. Proyecto de JPA con Hibernate

En la misma web de Hibernate se nos indica que si vamos a trabajar con JPA, incluyamos la dependencia hibernate-entitymanager, en lugar de hibernate-core (que por cierto, va incluida transitivamente en la anterior).

Si nos diera el siguiente error:

Java compiler level does not match the version of the installed Java project facet

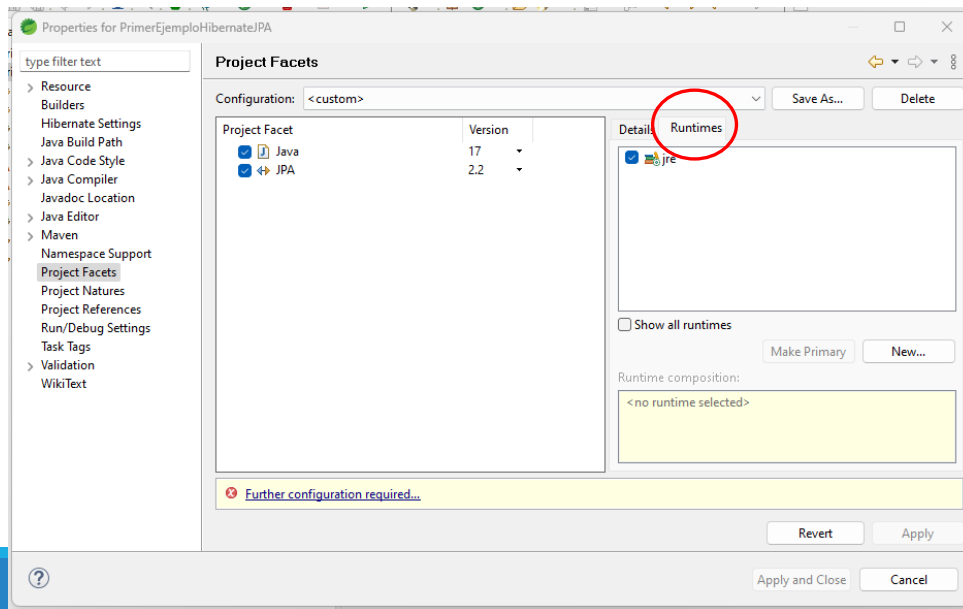
Para solucionarlo tendríamos que añadir las siguientes propiedades al archivo pom.xml:

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.target>1.8</maven.compiler.target>
    <maven.compiler.source>1.8</maven.compiler.source>
</properties>
```

En este caso, no es indispensable crear la carpeta */src/main/resources*.

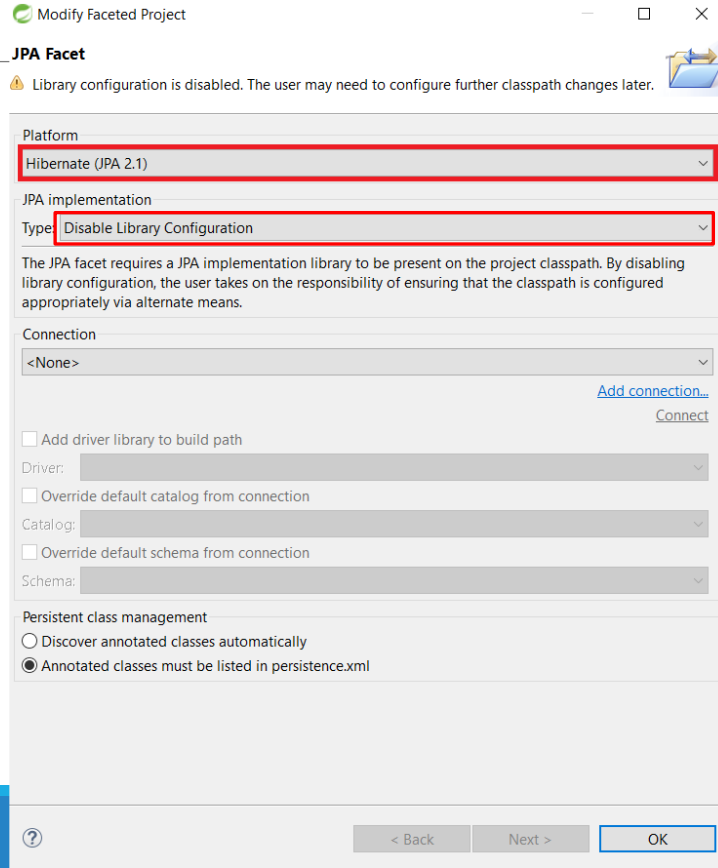
2. Proyecto de JPA con Hibernate

Este será el primer paso diferente al anterior. Pulsamos el botón derecho sobre el proyecto y seleccionamos la opción *Properties*. Seleccionamos *Project Facets*, convertimos el proyecto ([Covert to faceted form...](#)) y añadimos la característica JPA, y en la pestaña *Runtimes*, seleccionamos el JRE.



2. Proyecto de JPA con Hibernate

Pulsamos en *Further configuration required...* y los rellenamos con los siguientes datos:



Modify Faceted Project

JPA Facet

Library configuration is disabled. The user may need to configure further classpath changes later.

Platform
Hibernate (JPA 2.1)

JPA implementation
Type: Disable Library Configuration

The JPA facet requires a JPA implementation library to be present on the project classpath. By disabling library configuration, the user takes on the responsibility of ensuring that the classpath is configured appropriately via alternate means.

Connection
<None>

[Add connection...](#)
[Connect](#)

☐ Add driver library to build path

Driver:

☐ Override default catalog from connection

Catalog:

☐ Override default schema from connection

Schema:

Persistent class management

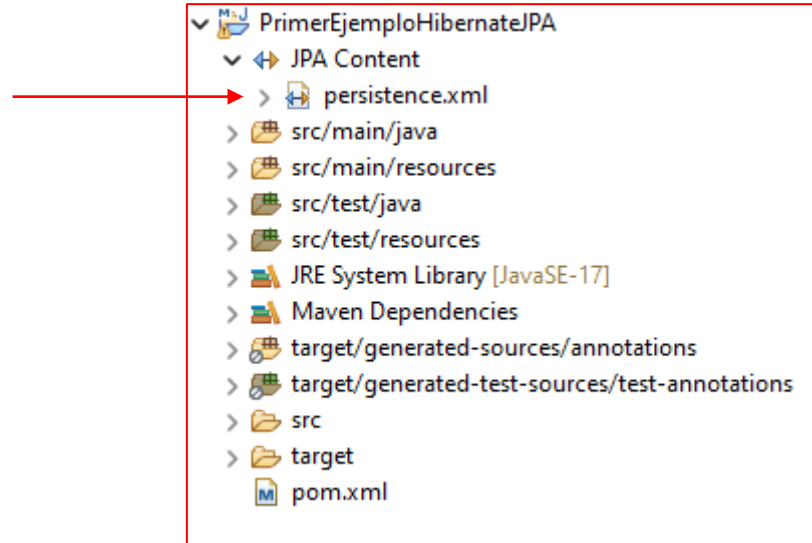
☐ Discover annotated classes automatically

☒ Annotated classes must be listed in persistence.xml

< Back Next > OK

2. Proyecto de JPA con Hibernate

A partir de ahora, nuestro proyecto incluye algunos elementos más, como la **unidad de persistencia**. Para verlo adecuadamente será necesario cambiar la perspectiva y abrir la de JPA



2. Proyecto de JPA con Hibernate

Antes de configurar la unidad de persistencia, vamos a añadir las entidades que vayamos a manejar en nuestro programa. Va a ser un simple clon del programa anterior, así que añadimos la misma clase User.

```
package com.iesvjp.hibernate.primerejemplohibernate;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class User {
    @Id
    private int id;
    @Column
    private String userName;
    @Column
    private String userMessage;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getUserName() {
        return userName;
    }

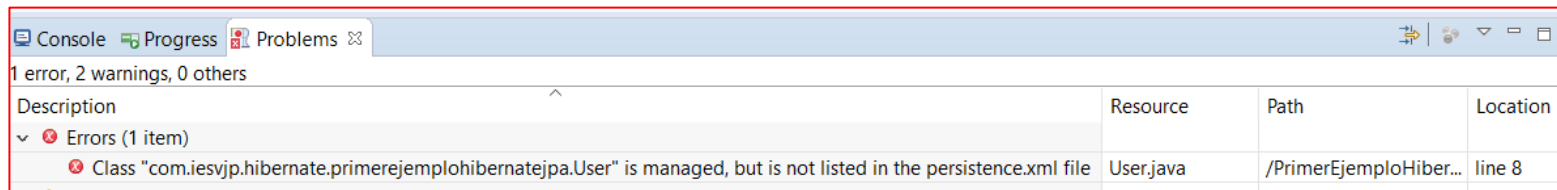
    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getUserMessage() {
        return userMessage;
    }

    public void setUserMessage(String userMessage) {
        this.userMessage = userMessage;
    }
}
```

2. Proyecto de JPA con Hibernate

Nada más añadirla, eclipse nos lanza un fallo, diciendo que por estar anotada, la clase es gestionada, pero que no está listada dentro de la unidad de persistencia. Vamos a solventarlo.

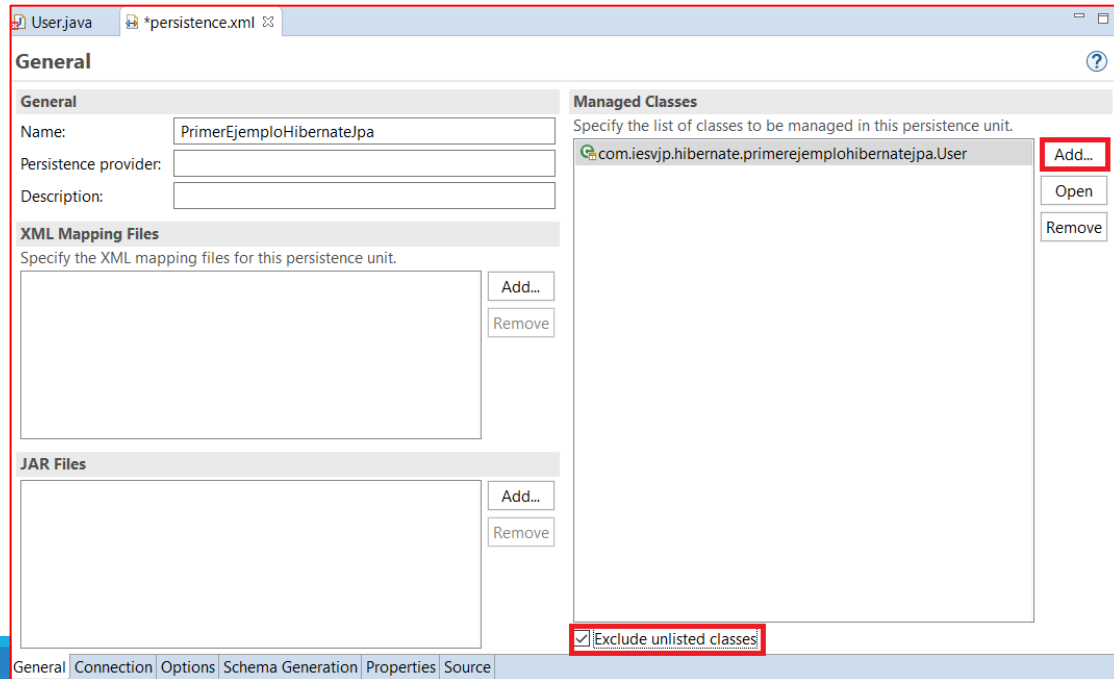


The screenshot shows the Eclipse IDE's 'Problems' view. The title bar indicates '1 error, 2 warnings, 0 others'. The table below lists the error details.

Description	Resource	Path	Location
✖ Errors (1 item)			
✖ Class "com.iesvjp.hibernate.primerejemplohibernatejpa.User" is managed, but is not listed in the persistence.xml file	User.java	/PrimerEjemploHiber...	line 8

3. Unidad de persistencia

Añadir la entidad como clase gestionada



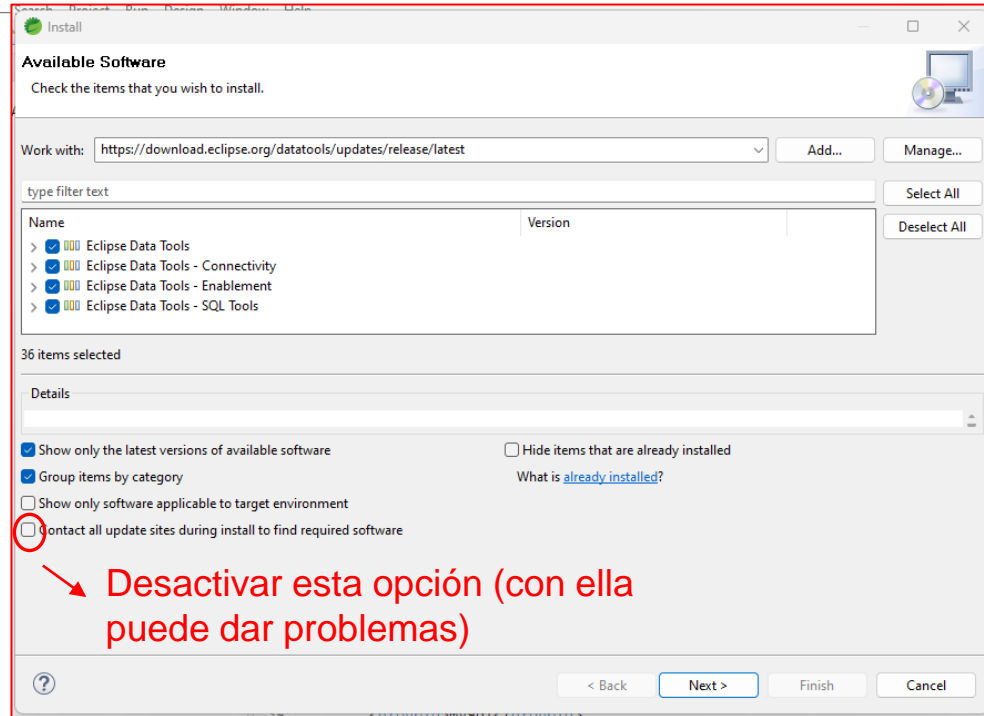
3. Unidad de persistencia

Para realizar la conexión vamos a utilizar el componente de **Eclipse Data Source Explorer**, si no estuviese instalado, nos los descargamos e instalamos.

Help → Install new Software

Y en la caja de texto Work with, añadimos la siguiente URL:

<https://download.eclipse.org/datatools/updates/release/1.15.0> (o /latest)



3. Unidad de persistencia

Posible solución de algunos errores:

<https://stackoverflow.com/questions/68809845/installing-new-software-in-spring>

“I found the answer by Binging the issue I was having. It turns out that de-selecting "Contact all update sites during install to find required software" improves the download performance. After that, the installation process finished in no time”

<https://stackoverflow.com/questions/70586854/resolving-plug-in-dependency-on-org-eclipse-equinox-p2-iu-in-eclipse-2021-12>

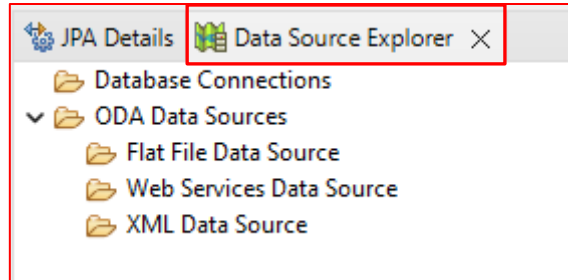
“Second solution: Add to "Available Software Sites" new site name: Eclipse Oxygen.2 URL: http://download.oracle.com/otn_software/oepe/library/eclipse-oxygen.2”

3. Unidad de persistencia

Para comprobar si hemos instalado correctamente el Eclipse Data Tools:

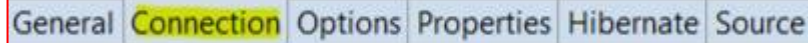
Window → Show View → Other...

Buscamos la opción “Data Source Explorer”. Nos saldrá una pestaña similar a esta:



3. Unidad de persistencia

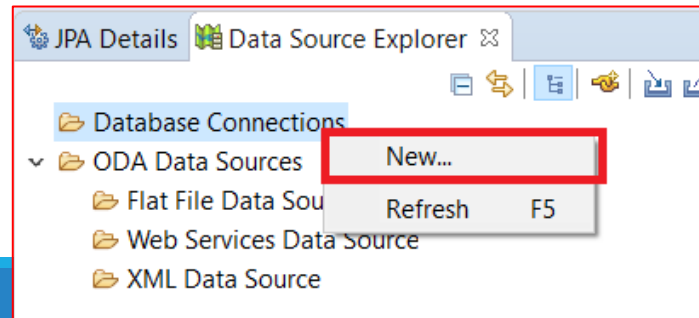
Después de instalar el plugin, volvemos al archivo de persistencia y nos vamos a la pestaña de *Connection*



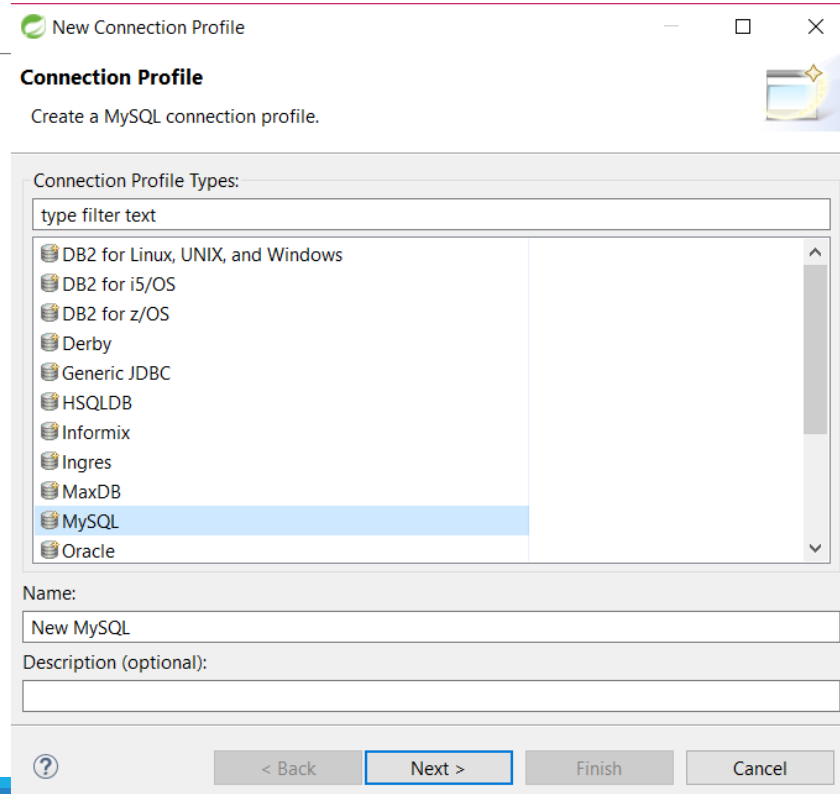
El tipo de transacción (Transaction Type) será de tipo Resource Local y, bien podemos establecer los datos a mano, o dar de alta una nueva conexión a base de datos, a través de la vista Data Source Explorer.

Vamos a crearnos una nueva conexión a través Data Source Explorer. Nos ponemos encima de Database Connections y pulsamos el botón derecho del ratón y seleccionamos New.

Seleccionamos MySQL y pulsamos Next



3. Unidad de persistencia

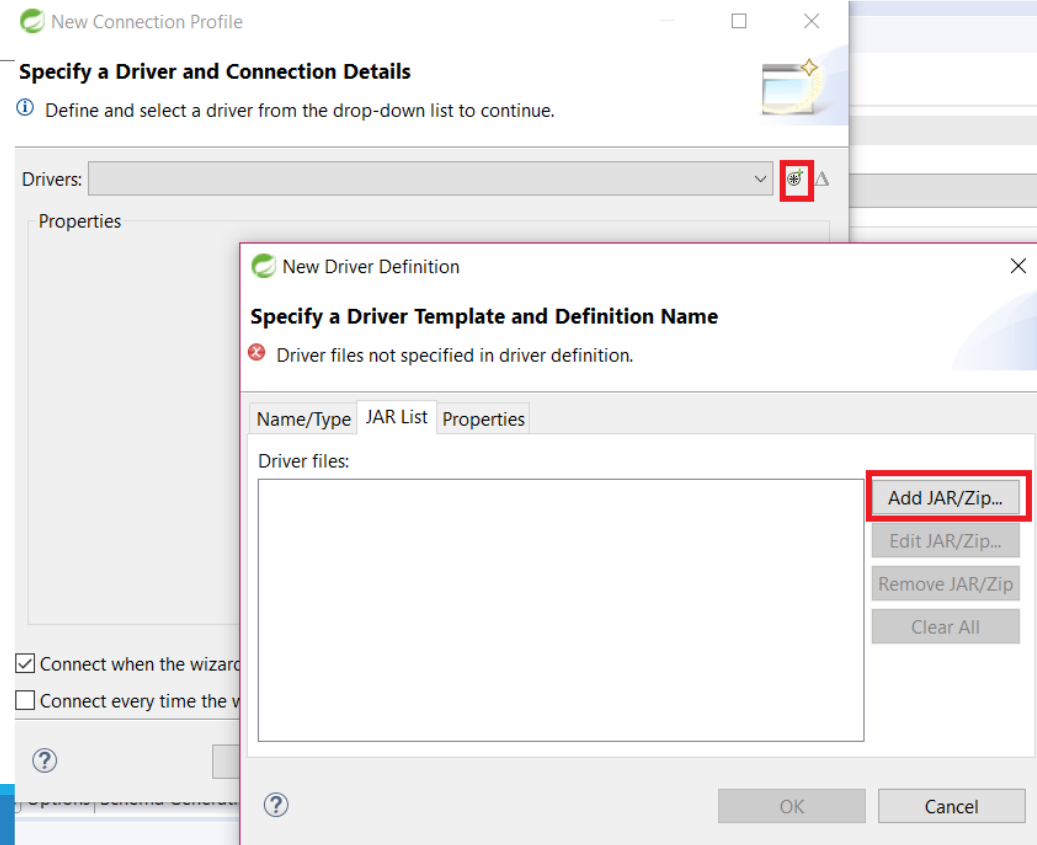


3. Unidad de persistencia

Pulsamos en el icono redondo para añadir los drivers y en la pestaña JAR List pulsamos el botón Add JAR/Zip y buscaremos el JAR del conector MySQL que hemos estado utilizando hasta ahora.

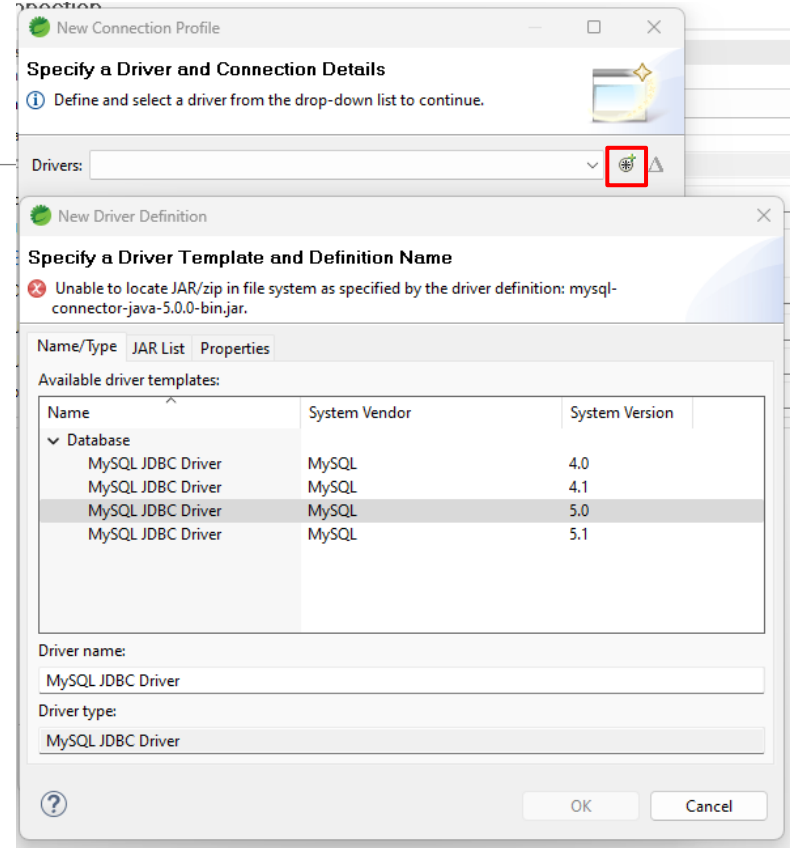
(mysql-connector-java-8.0.30)

Puedes encontrar también el JAR dentro de la carpeta de Maven del proyecto (siempre y cuando hayas instalado las dependencias previamente)



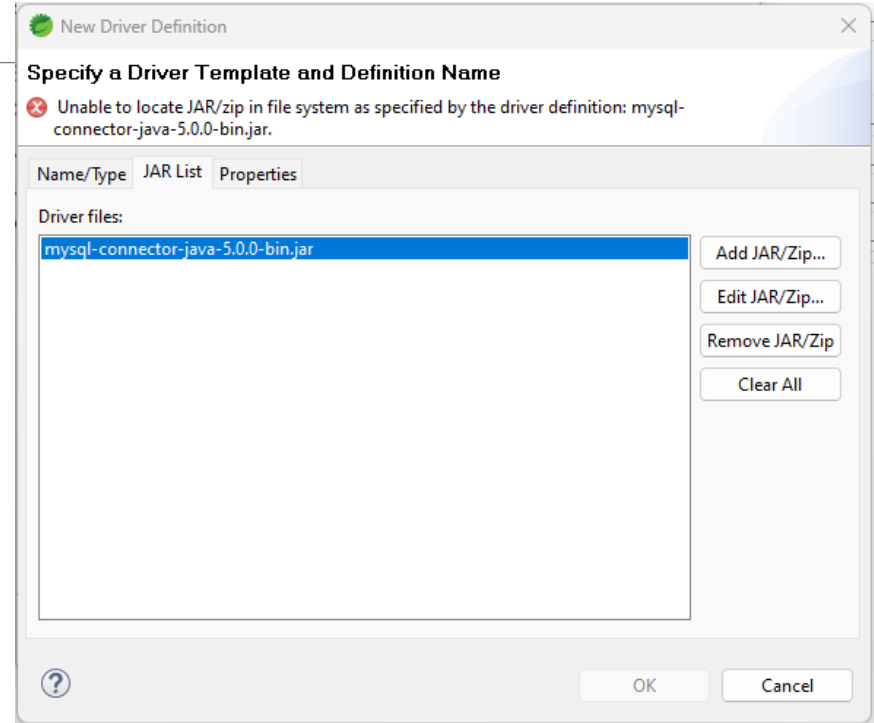
3. Unidad de persistencia

Es posible que la opción de añadir JAR esté inicialmente desactivada. Antes de activarla, podemos seleccionar la versión 5 de MySQL, y luego acceder a la pestaña de JARs.



3. Unidad de persistencia

Nos aparecerá un JAR por defecto que no encuentra. Ahora sí, a través de “Add JAR” añadimos el JAR de MySQL, y borramos el creado por defecto para que no dé problemas.

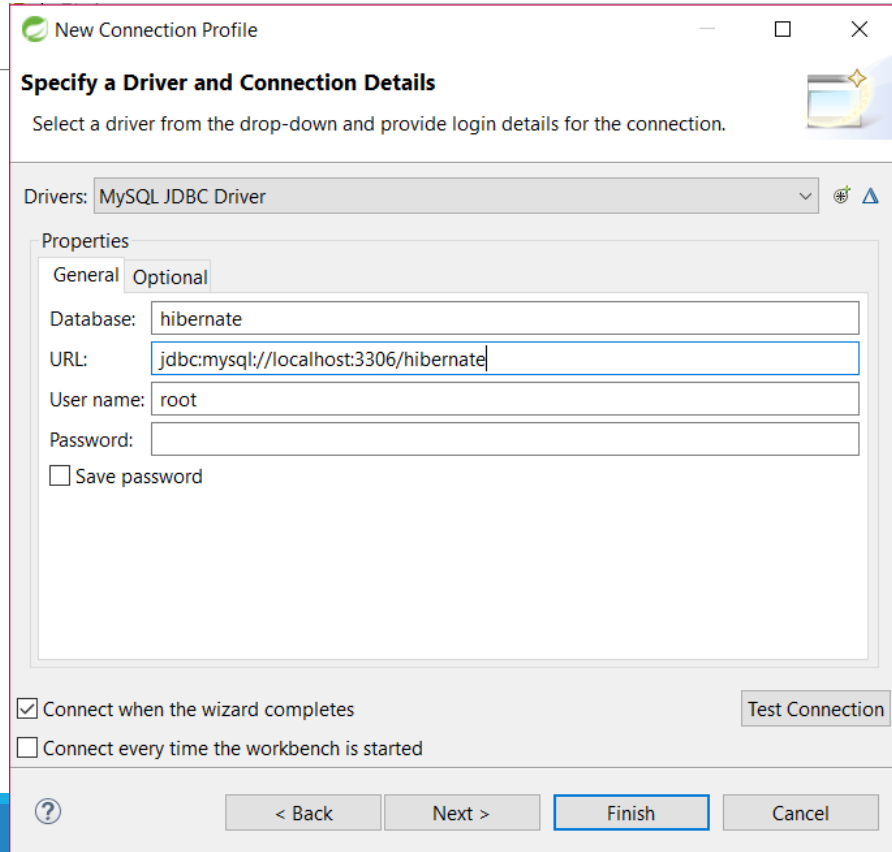


3. Unidad de persistencia

Buscamos la versión del driver y lo añadimos. Una vez añadido el driver, configuramos la conexión a nuestra base de datos llamada hibernate:

Pulsamos Next y Finish.

Importante: antes de pulsar finish, comprobar que el servidor MySQL está activo



New Connection Profile

Specify a Driver and Connection Details

Select a driver from the drop-down and provide login details for the connection.

Drivers: MySQL JDBC Driver

Properties

General Optional

Database: hibernate

URL: jdbc:mysql://localhost:3306/hibernate

User name: root

Password:

☐ Save password

☒ Connect when the wizard completes

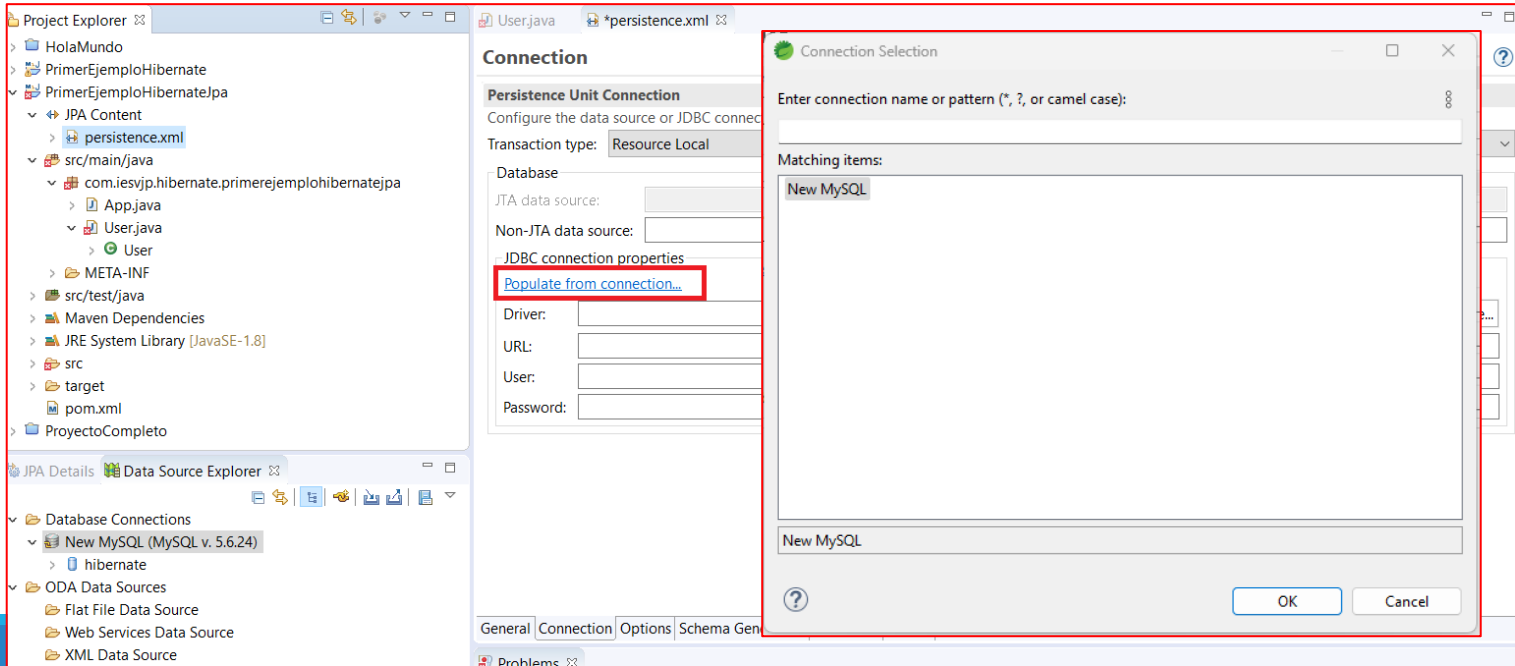
☐ Connect every time the workbench is started

Test Connection

? < Back Next > Finish Cancel

3. Unidad de persistencia

Una vez creada la conexión ya podemos seguir con la configuración de la unidad de persistencia, seleccionando la conexión a la BDD que acabamos de crear:



3. Unidad de persistencia

Vamos a establecer ahora algunos parámetros propios de Hibernate, para que nuestro proyecto pueda funcionar. Son muy parecidos al proyecto anterior:

- hibernate.dialect: org.hibernate.dialect.MySQL5InnoDBDialect
- hibernate.connection.driver: com.mysql.cj.jdbc.Driver
- hibernate.hbm2ddl.auto: create
- hibernate.show_sql: true
- hibernate.format_sql: true

Algunas se pueden añadir directamente desde la pestaña Hibernate, y otras desde la pestaña Properties.

persistence.xml

Properties

This table lists all properties that are defined for this persistence unit.

Name	Value
javax.persistence.jdbc.url	jdbc:mysql://localhost:3306/hibernate
javax.persistence.jdbc.user	root
javax.persistence.jdbc.password	
javax.persistence.jdbc.driver	com.mysql.cj.jdbc.Driver
hibernate.dialect	org.hibernate.dialect.MySQLInnoDBDialect
hibernate.connection.driver	com.mysql.cj.jdbc.Driver
hibernate.hbm2ddl.auto	create
hibernate.show_sql	true

Add...

Remove

4. Clase de aplicación

Ya podemos pasar a realizar nuestra clase de aplicación. Esta implementará las mismas tareas que en la lección anterior, pero la configuración inicial es algo diferente. Veamos el código:

4. Clase








```
1 package com.iesvjp.hibernate.primerejemplohibernatejpa;
2
3 import javax.persistence.EntityManager;
4 import javax.persistence.EntityManagerFactory;
5 import javax.persistence.Persistence;
6
7 /**
8  * Hello world!
9  *
10  */
11 public class App {
12     public static void main(String[] args) {
13         // Configuramos el EMF a través de la unidad de persistencia
14         EntityManagerFactory emf = Persistence.createEntityManagerFactory("PrimerEjemploHibernateJPA");
15
16         // Generamos un EntityManager
17         EntityManager em = emf.createEntityManager();
18
19         // Iniciamos una transacción
20         em.getTransaction().begin();
21
22         // Construimos un objeto de tipo User
23         User user1 = new User();
24         user1.setId(1);
25         user1.setUserName("Pepe");
26         user1.setUserMessage("Hello world from JPA with Pepe");
27
28         // Construimos otro objeto de tipo User
29         User user2 = new User();
30         user2.setId(2);
31         user2.setUserName("Juan");
32         user2.setUserMessage("Hello world from JPA with Juan");
33
34         // Persistimos los objetos
35         em.persist(user1);
36         em.persist(user2);
37
38         // Commiteamos la transacción
39         em.getTransaction().commit();
40
41         // Cerramos el EntityManager
42         em.close();
43         emf.close();
44     }
45 }
```

3. Unidad de persistencia

A diferencia del proyecto anterior, en este caso tenemos que inicializar dos objetos *EntityManagerFactory* y *EntityManager*. El segundo será nuestra interfaz directa con la base de datos, teniendo los métodos necesarios para consultar, actualizar, insertar o borrar datos. El primero es la factoría que nos permite construir al segundo, cargando los datos de nuestra unidad de persistencia.

Si **ejecutamos** la aplicación este sería el resultado en la BD:

+ Opciones

 				id	userMessage	userName
<input type="checkbox"/>		Editar		Copiar		Borrar
1	Hello world from JPA with Pepe				Pepe	
<input type="checkbox"/>		Editar		Copiar		Borrar
2	Hello world from JPA with Juan				Juan	

Recuerda que estamos ejecutando un proyecto Maven (clean, install, update project...)

Dudas y preguntas



Práctica

Como práctica de esta parte, realiza los mismos ejercicios de la Práctica 1, pero creando proyectos con JPA, tal y como hemos explicado en esta parte de la unidad.