

# Arquitectura de Software

- Diseño del más alto nivel de la estructura de un sistema
- Define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos.

## **Arquitectura multicapa:**

- Capa de presentación: Interfaz gráfica
- Capa de negocio: Programas
- Capa de datos: Bases de datos

**Acceso a datos:** Sirve datos de a la capa de negocio y *persiste* los datos que son enviados desde esta.

# Persistencia

- Capacidad de los datos o de la información de mantenerse de manera duradera a través del tiempo.
- Esta ha de ser garantizada incluso después de que el programa o sistema que los creó haya sido cerrado o reiniciado.
- Los programas tienen que transferir datos hacia y desde dispositivos de almacenamiento y deben proporcionar asignaciones desde las estructuras de datos del lenguaje de programación nativo a las estructuras de datos del dispositivo de almacenamiento.

# CRUD

Funciones básicas de la capa de persistencia

- **Create:** Agregar nuevas entradas
- **Read:** Recuperar información almacenada
- **Update:** Modificar una entrada existente
- **Delete:** Borrar una entrada

# Ejercicio: Persistencia con conectores

Conectores a BBDD mediante queries SQL.

- Trabaja con tipos de datos simples (enteros, reales, cadenas de texto, fechas...)
- La lógica de los programas trabaja con estructuras de datos complejos, comúnmente objetos.

## **Ejercicio:**

- Crear una capa de persistencia usando conectores
- Implementar las funciones CRUD
- La capa de persistencia debe manejar objetos de clases que representen la información almacenada en la BDD

# Gestión de proyectos (Maven)

- Automatización del sistema de construcción
- Ejemplo: Ant, Gradle, Maven...
- Gestión de dependencias
- Gestión del ciclo de vida del proyecto: compilación, test, empaquetado, instalación y despliegue
- Definición en el archivo POM.xml (Project Object Model)
- Integración con entornos de desarrollo

# Pruebas unitarias (JUnit)

- Garantizar la calidad del software mediante la validación de unidades individuales de código (métodos o funciones)
- Junit es la plataforma de referencia en Java
  - Los test se definen mediante el anotado `@Test`
  - Anotaciones adicionales importantes `@BeforeAll`, `@AfterAll`, `@BeforeEach`, `@AfterEach`...
  - Uso de métodos de aserción `assertEquals`, `assertTrue`, `assertAll`...
  - Uso de parámetros
- Cobertura de código: Verificar las partes del código que han sido probadas

# Pool de conexiones

Conjunto de conexiones de base de datos previamente establecidas y listas para ser reutilizadas

- Reducción del tiempo de conexión al reutilizar conexiones existentes.
- Controla el número máximo de conexiones simultáneas para evitar agotar recursos.
- Mejora el rendimiento al minimizar la sobrecarga asociada con la creación y destrucción de conexiones frecuentes.

La interfaz **DataSource** representa un pool de conexiones y existen varias implementaciones como HikariCP, Agroal y C3PO.

# Desfase Objeto-Relacional

- Diferencias conceptuales y de modelado entre los sistemas de gestión de bases de datos relacionales y los modelos de programación orientados a objetos.
- Las **bases de datos** almacenan datos simples en tablas y definen relaciones usando claves primarias y foráneas
- Los **programas** usan datos complejos en propiedades y métodos, además de manejar las relaciones mediante asociaciones entre objetos.
- Necesidad de traducción de un modelo al otro y viceversa
- Conciliación del modelo orientado a objetos con el modelo relacional de bases de datos.



# ORM

# Object-Relational Mapping

Mapeo relacional de objetos

# Mapeo objeto relacional

Conversión de datos entre el sistema de gestión de base de datos relacional y la representación de objetos del lenguaje de programación.

## Características:

- Abstracción de la base de datos.
- Eliminación de la necesidad de escribir consultas SQL directamente.
- Uso de funcionalidad propia de la orientación a objetos como la herencia y polimorfismo
- Portabilidad entre diferentes bases de datos
- Reutilización por varios sistema de explotación



# Objetivos

- Instalar herramientas ORM.
- Configurar herramientas ORM.
- Definir configuraciones de mapeo.
- Aplicar mecanismos de persistencia de objetos.
- Desarrollar aplicaciones de recuperación y modificación de objetos persistentes.
- Desarrollar aplicaciones que realicen consultas usando SQL.
- Gestionar transacciones.
- Desarrollar programas que utilicen bases de datos a través de herramientas ORM.

# Ventajas y Desventajas de los ORM

## Ventajas

- ✓ Aumenta de la productividad/rapidez de desarrollo
- ✓ Abstracción de la base de datos
- ✓ Reutilización en varios programas y/o bases de datos

## Desventajas

- ✗ Curva de aprendizaje elevada
- ✗ Configuración compleja
- ✗ Sobrecarga en el rendimiento

Los conectores permiten un control total del sistema subyacente y permiten una mayor optimización

# Herramientas mas usadas

- <https://github.com/topics/orm>
- <https://{language}.libhunt.com/>
- [https://en.wikipedia.org/wiki/List\\_of\\_object-relational\\_mapping\\_software](https://en.wikipedia.org/wiki/List_of_object-relational_mapping_software)
  - Hibernate (Java).
  - Entity Framework (C#).
  - SQLAlchemy (Python).

# JPA (Jakarta Persistence API)

- Especificación estándar para el mapeo objeto-relacional
- Define interfaces y anotaciones para el mapeo
- Proporciona un API estándar para realizar CRUD
- Varias implementaciones, como **Hibernate** o EclipseLink

# Hibernate

- Plataforma ORM para Java.
- Implementa y extiende JPA
- Sistemas de configuración estándar JPA y propio
- Herramientas para los IDE como HibernateTools dentro de JBoss Tools para eclipse
- <https://hibernate.org/orm/>

```
<!-- https://mvnrepository.com/artifact/org.hibernate.orm/hibernate-core -->  
<dependency>  
  <groupId>org.hibernate.orm</groupId>  
  <artifactId>hibernate-core</artifactId>  
  <version>6.4.2.Final</version>  
</dependency>
```

# Entidades

- Objetos que puede ser almacenados y recuperados de una base de datos relacional

## Anotaciones:

- **@Entity** marca una clase como entidad
- **@Id** marca un miembro como clave primaria
- Existen otras anotaciones para indicar tablas o columnas (@Table, @Column, @JoinColumn), relaciones (@ManyToOne @OneToMany...), etc.

> Las clases anotadas han de definir un **constructor vacío**

```
@Entity
public class Department
{
    @Id
    private int dept_no;
}
```



# Arranque de JPA/Hibernate

**EntityManagerFactory** (JPA) / **SessionFactory** (Hibernate)

- Instancia clases para gestión de la persistencia

**EntityManager** (JPA) / **Session** (Hibernate)

- Gestionan el ciclo de vida de las entidades
- Asociación con el *contexto* de persistencia
- Creación de queries y transacciones

# Configuración JPA

Archivo persistence.xml :

- Configuración central en proyectos JPA.
- Define:
  - Unidad de persistencia <persistence-unit>
  - Propiedades de conexión:
    - Driver, URL, usuario, contraseña...
- Debe estar en una directorio de nombre META-INF

# Instanciación

Creación:

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory(<nombre>);
```

Creación de EntityManager:

```
EntityManager em = emf.createEntityManager();
```

Operaciones de persistencia:

```
em.getTransaction().begin();  
em.persist(entidad);  
em.getTransaction().commit();
```

# Ejercicio Configuración e instanciación

- Crear un proyecto Maven
- Añadir las dependencias necesarias
- Crear un archivo persistence.xml
- Crear una instancia de EntityManager
- Obtener una transacción
- Persistir una entidad

# Gestión de transacciones

Aseguran que las operaciones de base de datos se realicen de manera consistente y exitosa o se deshagan en caso de error

- `getTransaction()`: Obtiene una transacción asociada a un contexto de persistencia
- `begin()`: Comienzo de la transacción
- `commit()`: Confirma los cambios
- `rollback()`: Deshace los cambios

```
try (var em = emf.createEntityManager()) {  
    EntityTransaction tx = em.getTransaction();  
    tx.begin();  
    // Operaciones  
    tx.commit();  
} catch (Exception e) {  
    tx.rollback();  
}
```

# EntityManager – CRUD

- Create: *void persist(Object entity)*
- Read: *<T> T find(Class<T> entityClass, Object primaryKey)*
- Delete: *void remove(Object entity)*
- Update:

# Contexto de Persistencia

Entorno donde se gestionan las entidades y su ciclo de vida.

Ciclo de vida de una entidad:

- Transient: La entidad nunca ha sido asociada a un contexto
- Persistent: La entidad está asociada a una BBDD mediante un contexto de persistencia
- Detached: La entidad ha dejado de estar asociada a un contexto
- Removed: La entidad está programada para ser eliminada

Características:

- Lazy/Eager Loading: Las asociaciones se cargan cuando se accede a ellas o de modo inmediato
- Gestión Automática de Cambios: El contexto de persistencia realiza un seguimiento automático de los cambios en las entidades pudiendo sincronizar estos explícitamente (flush)

# EntityManager: Funciones adicionales

**<T> T getReference(Class<T> entityClass, Object primaryKey) →**  
Obtiene una entidad con carga Lazy

**void detach(Object entity) →** Desconecta una entidad del contexto

**void refresh(Object entity) →** Recarga la información de la entidad desde la Base de Datos

**<T> T merge(T entity) →** Combina el estado de la entidad dada en el contexto de persistencia actual.

**void flush() →** Sincroniza el contexto de persistencia con la base de datos subyacente.



# Ejercicio: Relación entre EntityManager y sentencias SQL

- Configurar e instanciar un EntityManager
- Configurar la salida de Sql mediante las propiedades show/format/highlight(\_sql)
- Realizar multiples combinaciones de operaciones find/persist/remove/getReference...
- Observar las sentencias sql que se producen
- Encontrar situaciones donde las combinaciones de las funciones de persistencia no tienen asociada una sentencia SQL

# Anotaciones: Id, table & column

- @Table(**name**, schema, catalog)
- @Id
  - @GeneratedValue(**strategy**)
    - SEQUENCE, IDENTITY, TABLE, UUID
- @IdClass; @Embeddable y @EmbeddedId
  - Implementar *Serializable* y sobrescribir *Equals* y *HashCode*
- @Column(**name**, nullable, length)

# Ejercicio

- Usando una base de datos de prueba con una tabla departamentos
  - Probar las diferentes estrategias para la creación de Id
- Añadir nuevas columnas a la tabla departamento como presupuesto
- Modificar la tabla para que la clave primaria sea el código postal y el nombre
  - Probar a crear el Id usando ambos IdClass y EmbeddedId
- Las entidades definidas han de tener nombre de clase y campos en inglés

# Ejercicio

- Modelar un conjunto de "productos"
- Los productos tienen al menos las siguientes características: "nombre", "precio", "iva". Siendo el nombre una cadena de caracteres y el precio y el iva números reales.
- Crear una entidad "Product" con los campos "name", "price" y "vat" que mapee la tabla productos.
- Usando JPA implementar una clase que realice las siguientes funciones
  - Añadir productos, usando el precio con iva y el iva
  - Modificar el precio de un producto en un porcentaje dado
  - Comprobar si un producto tiene un precio menor que un valor dado y eliminarlo de la tabla en caso positivo
- Comprobar las diferencias entre las diferentes estrategias de la generación de identificadores.

# Relaciones

Establecimiento de relaciones entre entidades, reflejando la relación subyacente en la base de datos

- @OneToOne, @ManyToOne, @OneToMany, **@ManyToMany**
  - @JoinColumn, **@JoinTable**

Las relaciones pueden ser unidireccionales o bidireccionales gracias al uso del parámetro **mappedBy**

El parámetro **cascade** permite que las operaciones del contexto sean realizadas en las entidades con la que se tiene relación.

El parámetro **fetch** permite configurar la lectura eager/lazy

El lado “Many” puede almacenar las entidades en varios tipos de **colecciones**: Set, List...

# Ejercicio

- Definir la entidad empleado, así como la relación ManyToOne
- Hacer la relación bidireccional
- Crear una tabla *estación de trabajo* y su entidad con una relación OneToOne con un empleado
- Definir una tabla proyecto que esté relacionado con los empleados de modo ManyToMany

# Ejercicio

- Modelar los datos relacionados con un Centro de Formación de Profesional.
- Queremos controlar la información referente a los ciclos, los módulos, los profesores, los alumnos y los expedientes.
- Las relaciones entre los datos son las siguiente:
  - Un ciclo se compone de varios módulos
  - Los alumnos solo pueden estar matriculados en un ciclo
  - Cada alumno tiene un expediente asociado
  - Un profesor puede dar clase en varios módulos
  - Un módulo puede ser impartido por varios profesores
- Realizar varias operaciones con los datos:
  - Añadir varios ciclos, módulos, profesores y centros.
  - Eliminar un ciclo y por consiguiente sus módulos.
  - Obtener las notas de todos los alumnos de un módulo

# Consultas con JPQL

JPQL (Java Persistence Query Language)

- Orientado a objetos
- Similar a SQL (SELECT <> FROM <> WHERE <>)
- Permite Select, Update, Remove. Pero no Inserts
- Uso de parámetros
- TypedQuery and DTO
  - Data Transfer Object (record)
  - Select NEW dto.Result(a, b)
- @NamedQuery

```
TypedQuery<Entidad> query = em.createQuery("""  
    SELECT e  
    FROM Entidad e  
    WHERE e.propiedad = :valor  
    """,  
    Entidad.class);  
query.setParameter("valor", valor);  
List<Entidad> resultados = query.getResultList();
```



# Consultas con SQL Nativo

## SQL Nativo

```
Query query = em.createNativeQuery("SELECT * FROM tabla WHERE columna = ?", Entidad.class);  
query.setParameter(1, valor);  
List<Entidad> resultados = query.getResultList();
```

- Uso de parámetros
- Mapeo a entidades
- Uso de Procedimientos

# Ejercicio

- Consultas JPQL
  - Todos los empleados de un departamento dado.
  - Nombre del departamento que tiene la mayor cantidad de empleados.
  - Empleados con salario superior al promedio:
  - Proyectos en los que participa un empleado específico
- Consultas NativeSQL
  - Empleados por rango salarial:
  - Nombre del departamento con más proyectos:
  - Empleados de un determinado proyecto y departamento
  - Cantidad de empleados por departamento:
  - Empleados con mayor antigüedad en la empresa

# Query Criteria

Construcción de consultas de manera programática y tipada

Creación

- `CriteriaBuilder builder = entityManager.getCriteriaBuilder();`
- `CriteriaQuery<Entidad> criteriaQuery = builder.createQuery(Entidad.class);`

Defunción de Root y predicado

- `Root<Entidad> root = criteriaQuery.from(Entidad.class);`
- `Predicate condicion = builder.equal(root.get("atributo"), valor);`
- `criteriaQuery.where(condicion);`

Ejecución y resultados

- `TypedQuery<Entidad> typedQuery = entityManager.createQuery(criteriaQuery);`
- `List<Entidad> resultados = typedQuery.getResultList();`

# Ejercicio JPQL and Query Criteria

- Todos los departamentos
- Todos los empleados de un departamento
- Empleados que tienen un salario entre dos valores dados
- Empleados que participan en un proyecto específico
- Departamentos que tienen al menos un empleado con un salario superior a cierta cantidad
- Cantidad total de empleados en cada departamento

# Hibernate Tool de JBoss

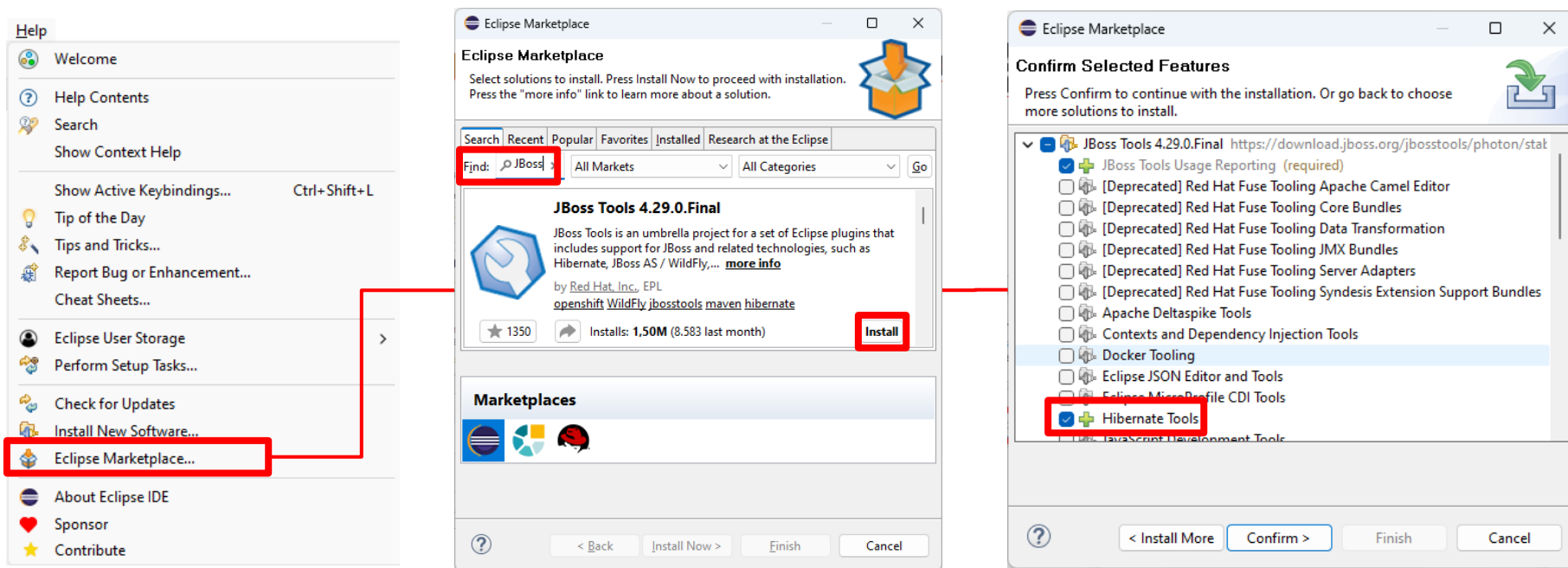
- Help > Eclipse Marketplace
- Buscar “JBoss Tools” y seleccionar “Install”
- Deseleccionar todo y seleccionar “Hibernate Tools”
- Instalar (Next, Trust...)

Comprobar la instalación

- Windows > Preference > Jboss Tools > Hibernate

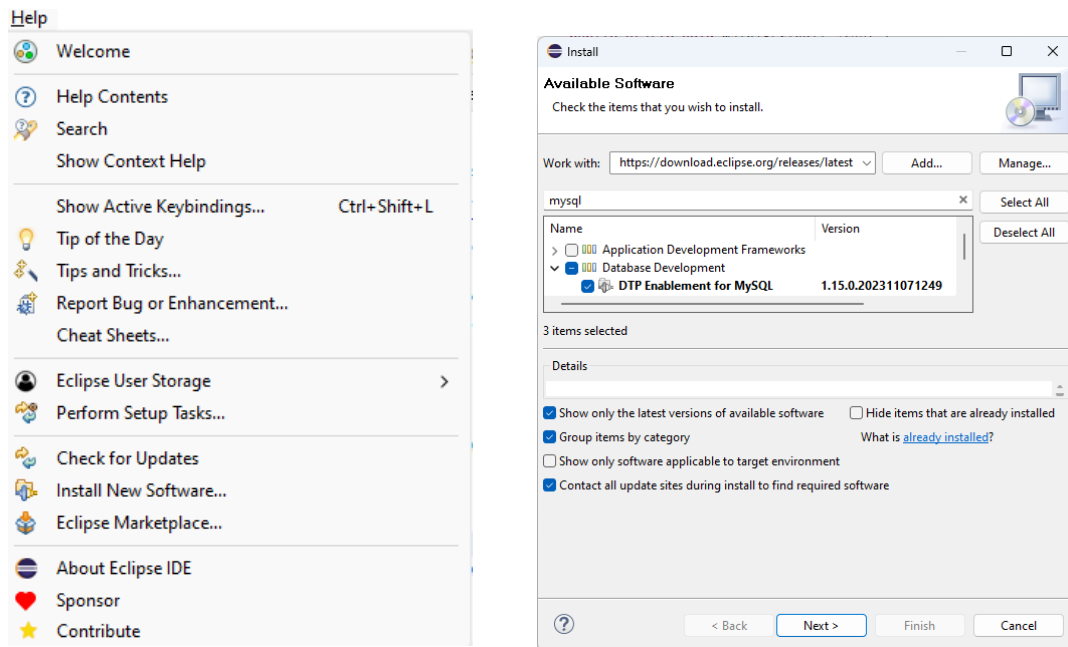
# Hibernate Tool de JBoss

Help > Eclipse Marketplace » Jboss » Hibernate Tools



# Install Database Development

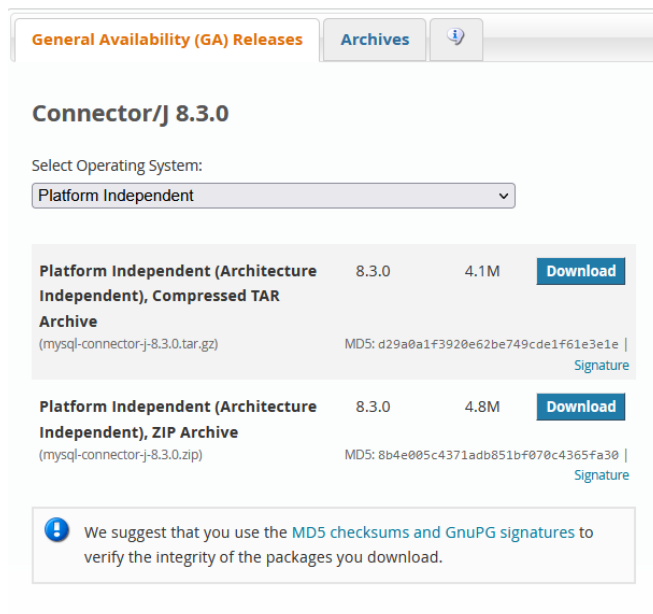
- Help > Install New Software...
- Latest Eclipse Simultaneous Release - <https://download.eclipse.org/releases/latest>



- DTP Connectivity
- DTP Enablement for MySQL
- DTP ModelBase

# Download MySQL JDBC connector

- <https://dev.mysql.com/downloads/connector/j/>



General Availability (GA) Releases Archives ⓘ

## Connector/J 8.3.0

Select Operating System:

Platform Independent ▾

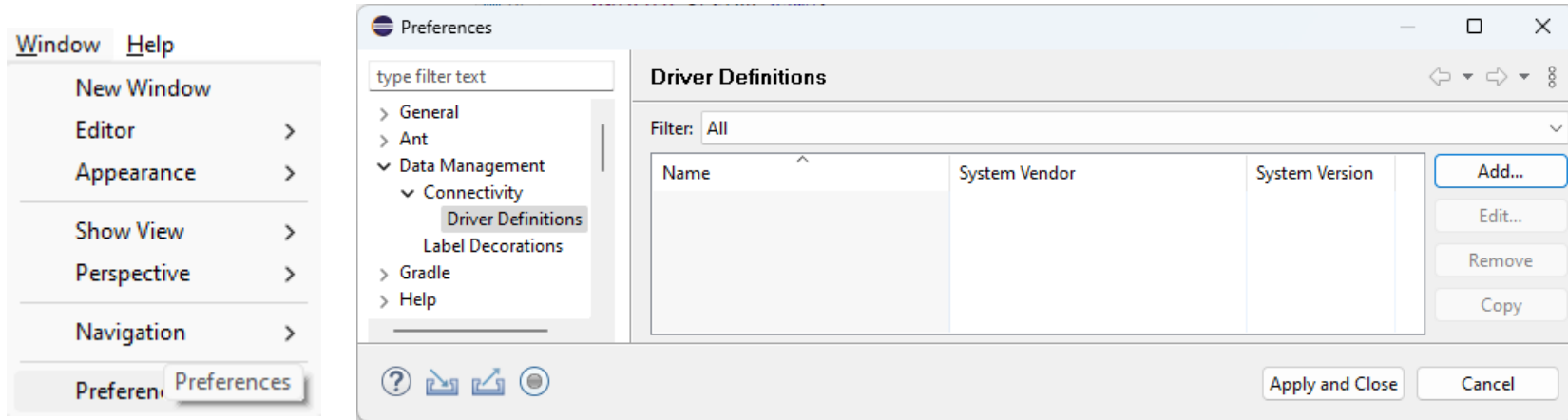
<b>Platform Independent (Architecture Independent), Compressed TAR Archive</b> (mysql-connector-j-8.3.0.tar.gz)	8.3.0	4.1M	<a href="#">Download</a>
MD5: d29a0a1f3920e62be749cde1f61e3e1e   <a href="#">Signature</a>			
<b>Platform Independent (Architecture Independent), ZIP Archive</b> (mysql-connector-j-8.3.0.zip)	8.3.0	4.8M	<a href="#">Download</a>
MD5: 8b4e005c4371adb851bf070c4365fa30   <a href="#">Signature</a>			

ⓘ We suggest that you use the [MD5 checksums](#) and [GnuPG signatures](#) to verify the integrity of the packages you download.



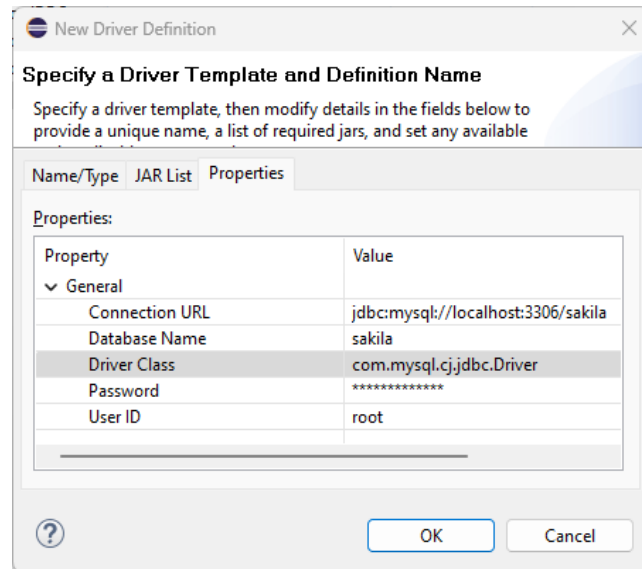
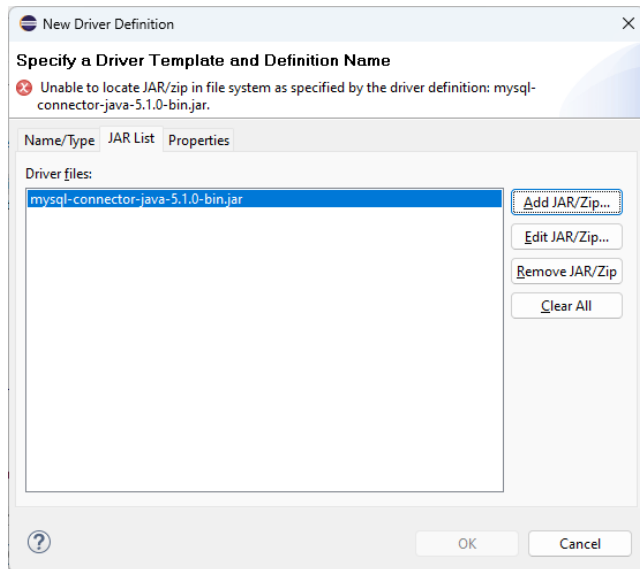
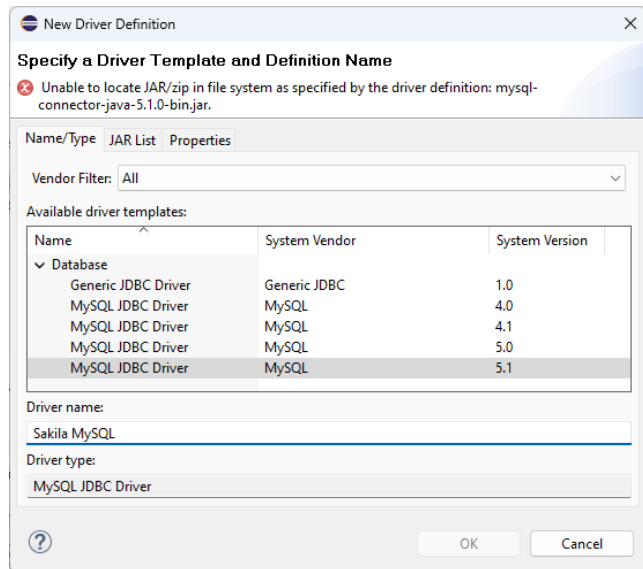
# Add Driver Definition

- Window > Preferences > Data Management > Connectivity > Driver Definitions » Add...



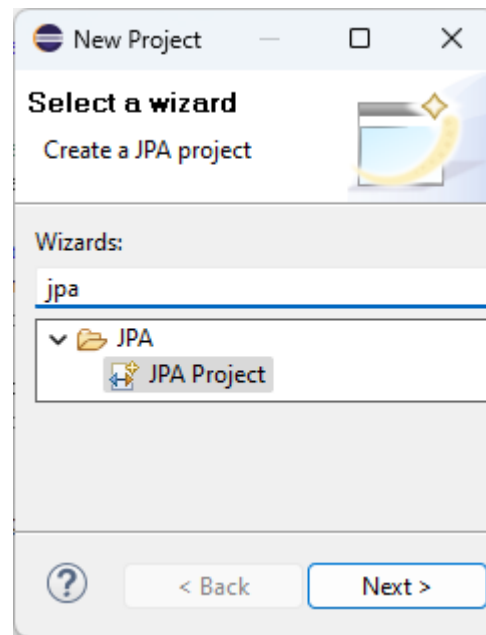
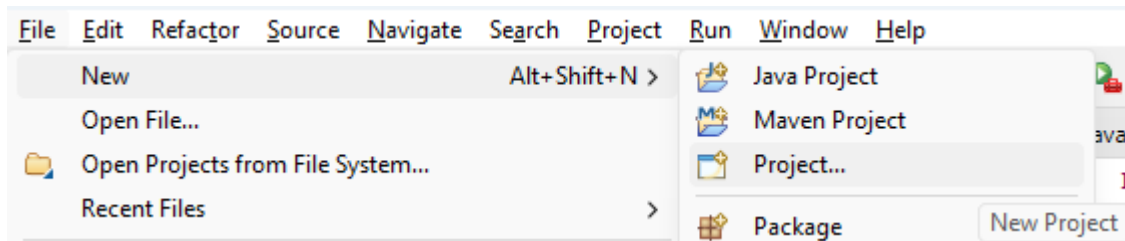
# New Driver Definition

- MySQL 5.1> Driver Name
- Remove JAR / Add JAR (Downloaded MySQL JDBC Connector)
- Properties: com.mysql.cj.jdbc.Driver...



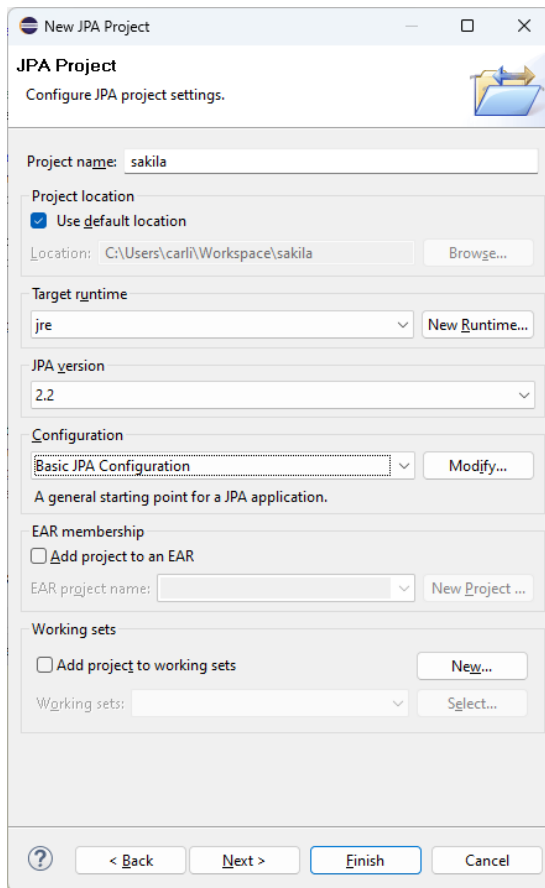
# New JPA Project

File > New > Project... » JPA Project



# New JPA Project

- Jre
- JPA 2.2
- Basic JPA



New JPA Project

JPA Project  
Configure JPA project settings.

Project name: sakila

Project location  
☒ Use default location  
Location: C:\Users\carli\Workspace\sakila Browse...

Target runtime  
jre New Runtime...

JPA version  
2.2

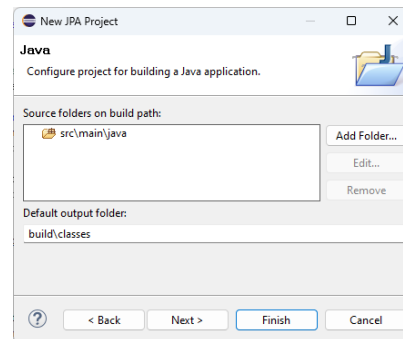
Configuration  
Basic JPA Configuration Modify...

A general starting point for a JPA application.

EAR membership  
☐ Add project to an EAR  
EAR project name: New Project ...

Working sets  
☐ Add project to working sets New...  
Working sets: Select...

? < Back Next > Finish Cancel



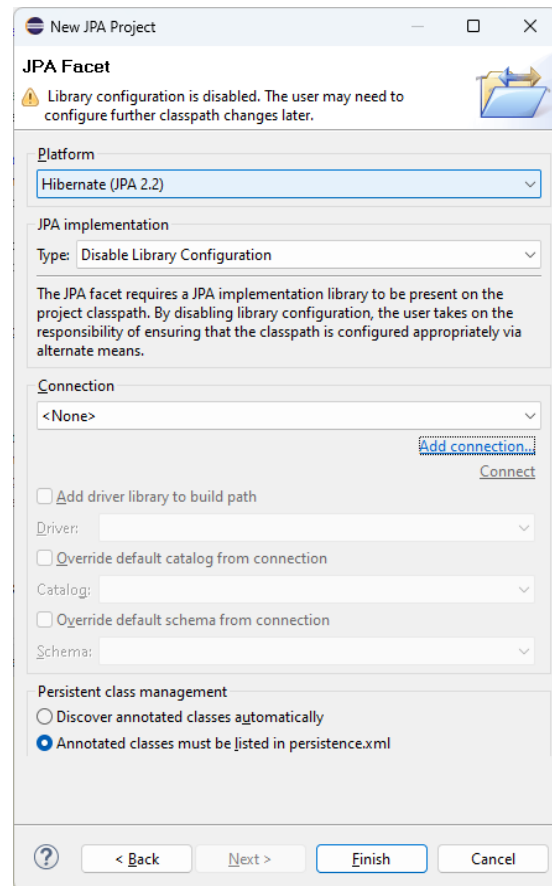
New JPA Project

Java  
Configure project for building a Java application.

Source folders on build path:  
src/main/java Add Folder... Edit... Remove

Default output folder:  
build/classes

? < Back Next > Finish Cancel



New JPA Project

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

Platform  
Hibernate (JPA 2.2)

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

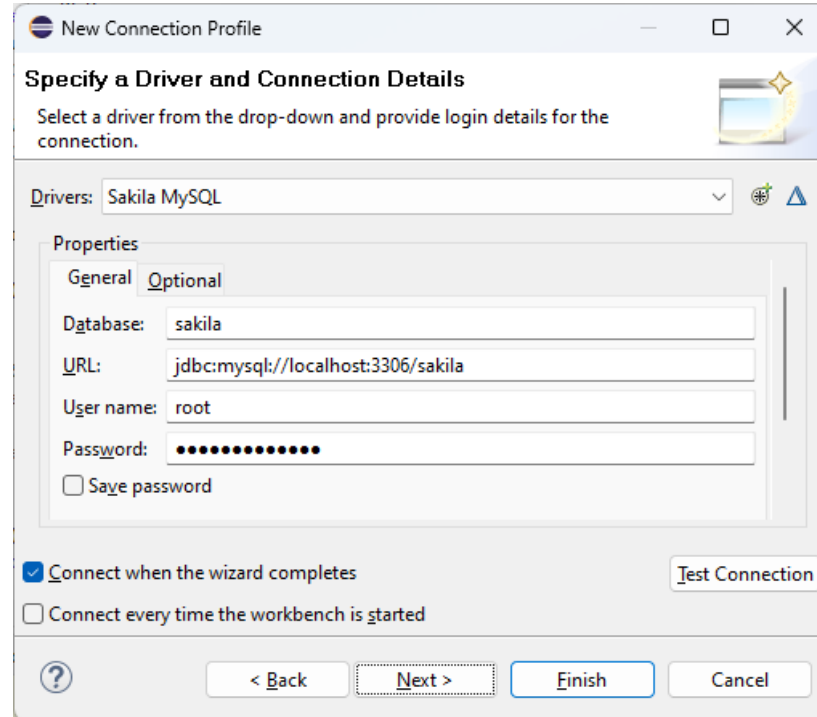
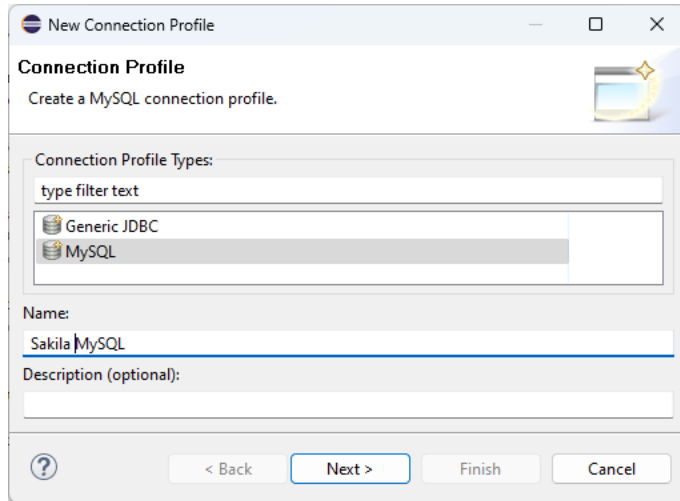
☐ Add driver library to build path  
Driver:   
☐ Override default catalog from connection  
Catalog:   
☐ Override default schema from connection  
Schema:

Persistent class management  
☐ Discover annotated classes automatically  
☒ Annotated classes must be listed in persistence.xml

? < Back Next > Finish Cancel

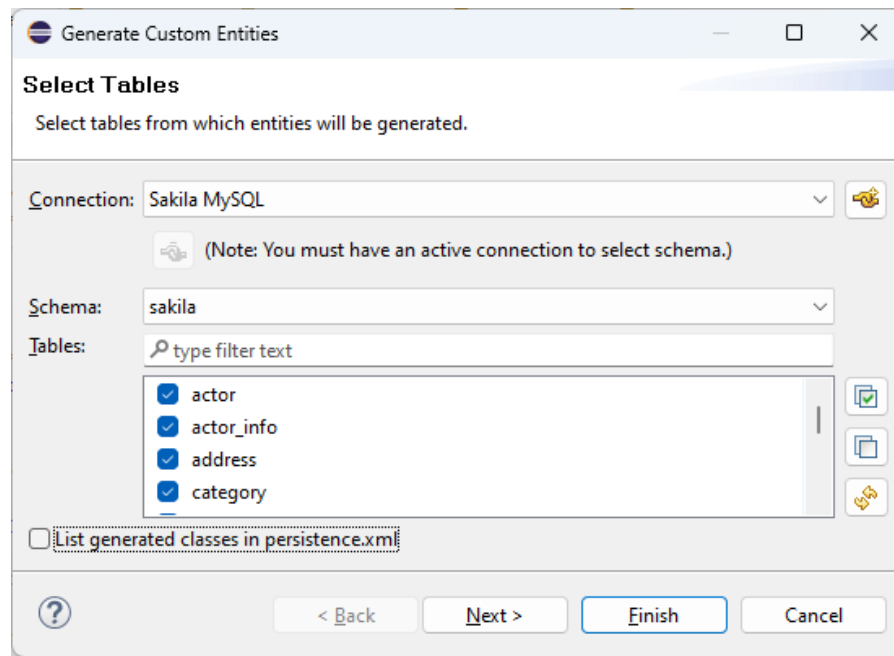
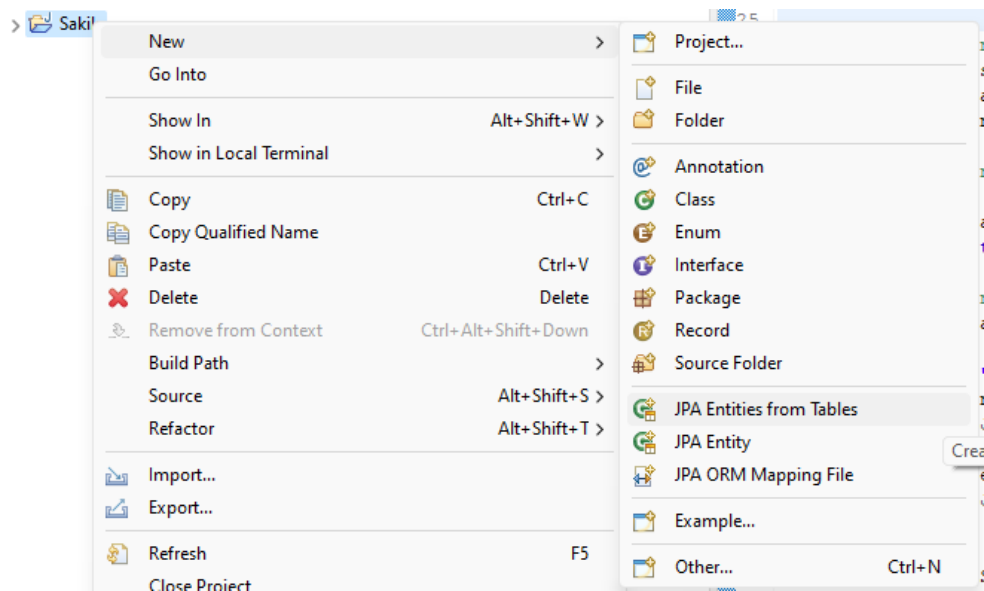
# Add Connection

- MySQL



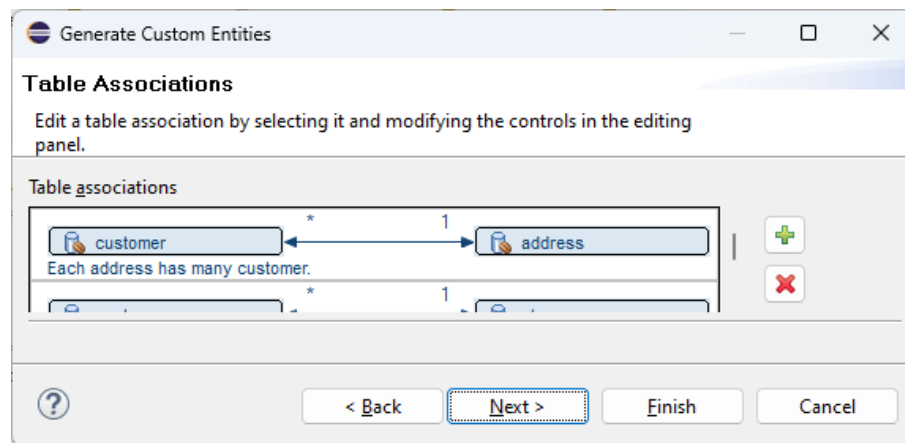
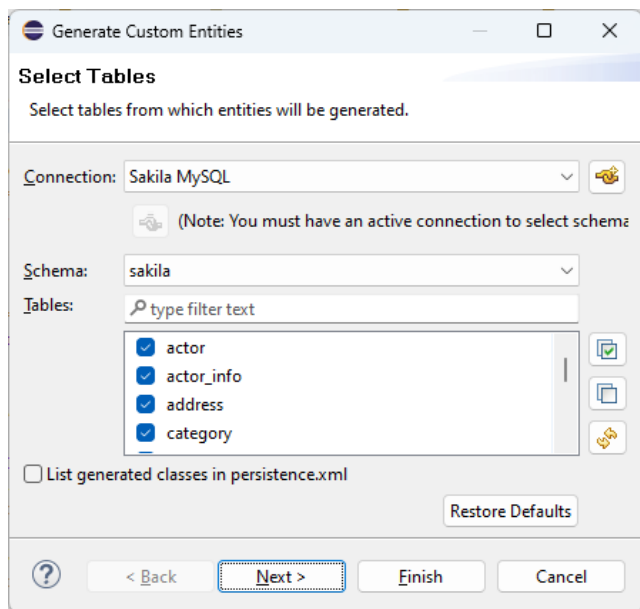
# Add Connection

- New > JPA Entities from Tables



# Entities & Relation

- Deseleccionar tablas intermedias
- Modificar relaciones



# Many To Many

- New > JPA Entities from Tables

Create New Association

Association Tables

Specify the association tables.

Association kind

☐ Simple association

☒ Many to many association

Association tables:

Table 1: actor

Table 2: film

Join table: film\_actor

< Back Next > Finish Cancel

Create New Association

Association Cardinality

Specify the association cardinality.

☐ Many to one

Each film has many actor.

☐ One to many

Each actor has many film.

☐ One to one

There is one actor per film.

☒ Many to many

Each actor has many film, and each film has many actor.

< Back Next > Finish Cancel

Create New Association

Join Columns

Specify the join columns.

Specify the join columns between the actor and film\_actor tables:

actor	film_actor

Add Remove

Specify the join columns between the film\_actor and film tables:

film_actor	film

Add Remove

< Back Next > Finish Cancel



# Customize

- Key Generatoes
- Enity access
- Fetch
- Collections

The screenshot shows a Java Swing dialog box titled "Generate Custom Entities". The "Customize Defaults" tab is selected. The dialog allows users to customize the default settings for generating entities. It includes sections for "Mapping defaults" (key generator, sequence name, entity access, associations fetch, collection properties type) and "Domain java class" (source folder, package, superclass, interfaces). The "Next >" button is highlighted.

**Generate Custom Entities**

**Customize Defaults**

Optionally customize aspects of entities that will be generated by default from database tables. A Java package should be specified.

**Mapping defaults**

Key generator: none

Sequence name:

You can use the patterns \$table and/or \$pk in the sequence name. These patterns will be replaced by the table name and the primary key column name when a table mapping is generated.

Entity access: ☒ Field ☐ Property

Associations fetch: ☒ Default ☐ Eager ☐ Lazy

Collection properties type: ☐ java.util.Set ☒ java.util.List

☐ Always generate optional JPA annotations and DDL parameters

**Domain java class**

Source folder: Sakila/src/main/java

Package: model

Superclass:

Interfaces: ☒ java.io.Serializable

# Conocimientos

- **Concepto de mapeo objeto relacional.**
- **Características de las herramientas ORM. Herramientas ORM más utilizadas.**
- **Instalación de una herramienta ORM. Configuración.**
- **Estructura de un fichero de mapeo. Elementos, propiedades.**
- **Mapeo basado en anotaciones.**
- **Clases persistentes.**
- **Sesiones; estados de un objeto.**
- **Carga, almacenamiento y modificación de objetos.**
- **Consultas SQL.**
- **Gestión de transacciones.**
- **Desarrollo de programas que utilizan bases de datos a través de herramientas ORM.**

# Referencias

- [https://docs.jboss.org/hibernate/orm/6.4/quickstart/html\\_single](https://docs.jboss.org/hibernate/orm/6.4/quickstart/html_single)
- [https://docs.jboss.org/hibernate/orm/6.4/introduction/html\\_single/Hibernate\\_Introduction.html](https://docs.jboss.org/hibernate/orm/6.4/introduction/html_single/Hibernate_Introduction.html)
- [https://docs.jboss.org/hibernate/orm/6.4/userguide/html\\_single/Hibernate\\_User\\_Guide.html](https://docs.jboss.org/hibernate/orm/6.4/userguide/html_single/Hibernate_User_Guide.html)
- <https://jakarta.ee/specifications/persistence/3.0/jakarta-persistence-spec-3.0.html>