



**NetBeans**

# Configuración JPA en Netbeans

Ronald Cuello

# Prerrequisitos

Para realizar este ejemplo sencillo ,voy a utilizar el IDE Netbeans versión 6.8 por que es la versión que tengo instalada en mi maquina, aunque también pueden realizar esta configuración en versiones 6.0 hacia arriba.

Y como motor de base de datos ,MySQL

Listo empecemos a configurar JPA!!!

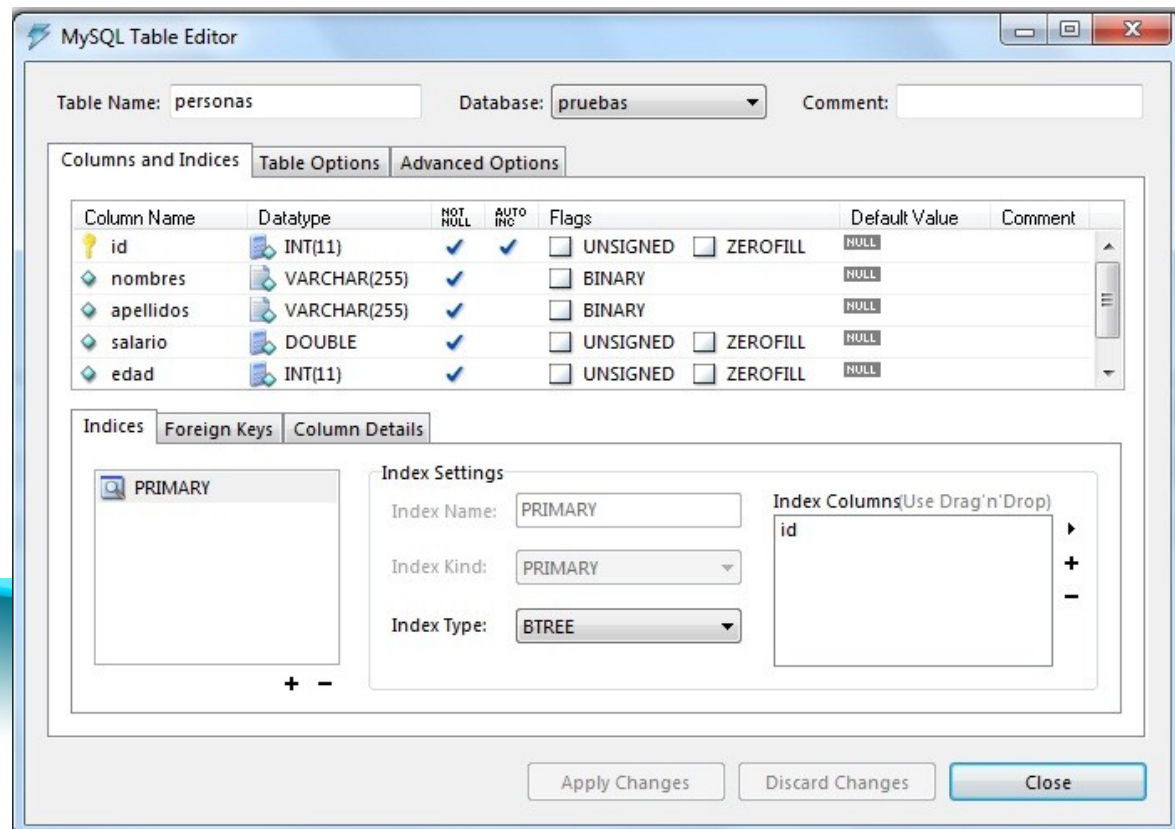




# Creación Base de datos

Para nuestro ejemplo, se necesita crear una base de datos de prueba, y una tabla llamada **PERSONAS**.

jdbc:mysql://localhost:3306/pruebas [root on Default schema]



MySQL Table Editor

Table Name:  Database:  Comment:

Columns and Indices | Table Options | Advanced Options

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
id	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL	<input type="text" value="NULL"/>	
nombres	VARCHAR(255)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> BINARY	<input type="text" value="NULL"/>	
apellidos	VARCHAR(255)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> BINARY	<input type="text" value="NULL"/>	
salario	DOUBLE	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL	<input type="text" value="NULL"/>	
edad	INT(11)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL	<input type="text" value="NULL"/>	

Indices | Foreign Keys | Column Details

PRIMARY

Index Settings

Index Name:

Index Kind:

Index Type:

Index Columns (Use Drag'n'Drop)

id

Apply Changes Discard Changes Close

# Código SQL para crear la tabla

```
DROP TABLE IF EXISTS `pruebas`.`personas`;
```

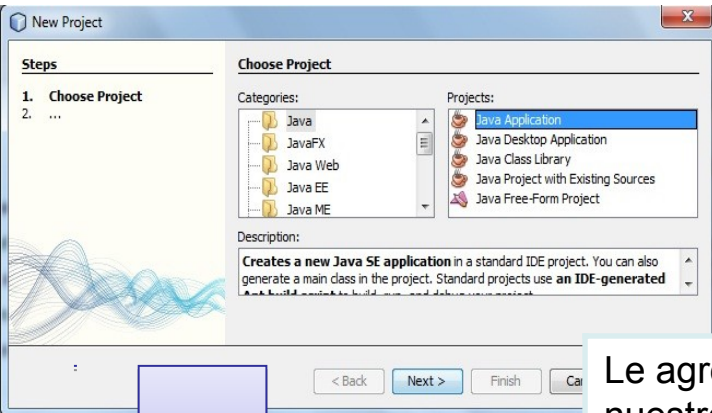
```
CREATE TABLE `pruebas`.`personas`  
( `id` int(11) NOT NULL AUTO_INCREMENT,  
  `nombres` varchar(255) NOT NULL,  
  `apellidos` varchar(255) NOT NULL,  
  `salario` double NOT NULL,  
  `edad` int(11) NOT NULL, PRIMARY KEY (`id`)  
)
```

```
ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1;
```



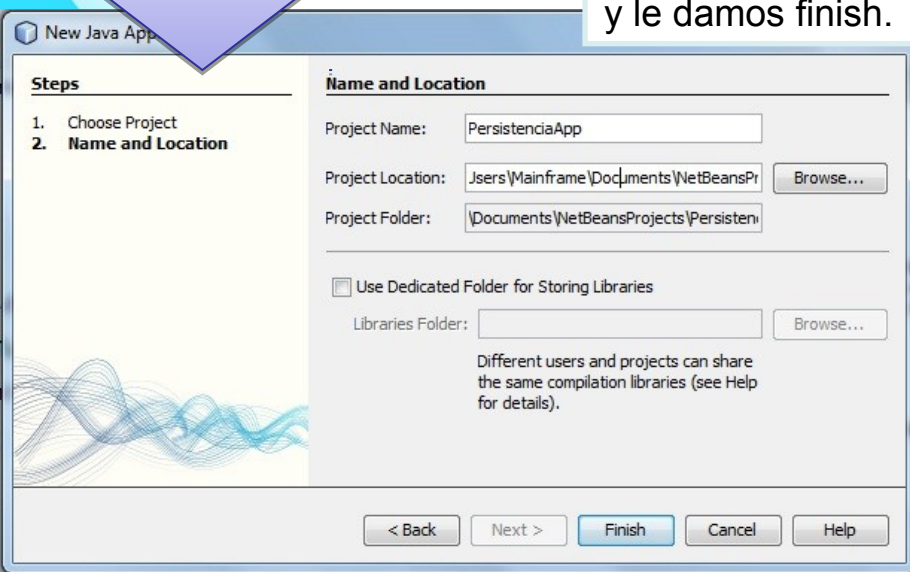


# Creando el Proyecto

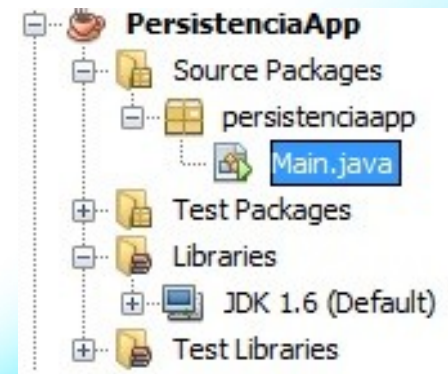


Seleccionamos como plantilla “**Java Application**” para realizar nuestra consola de pruebas

Le agregamos un nombre a nuestro proyecto, en este caso será “**PersistenciaApp**” y le damos finish.



Archivos generados por el IDE.  
Nada raro aun

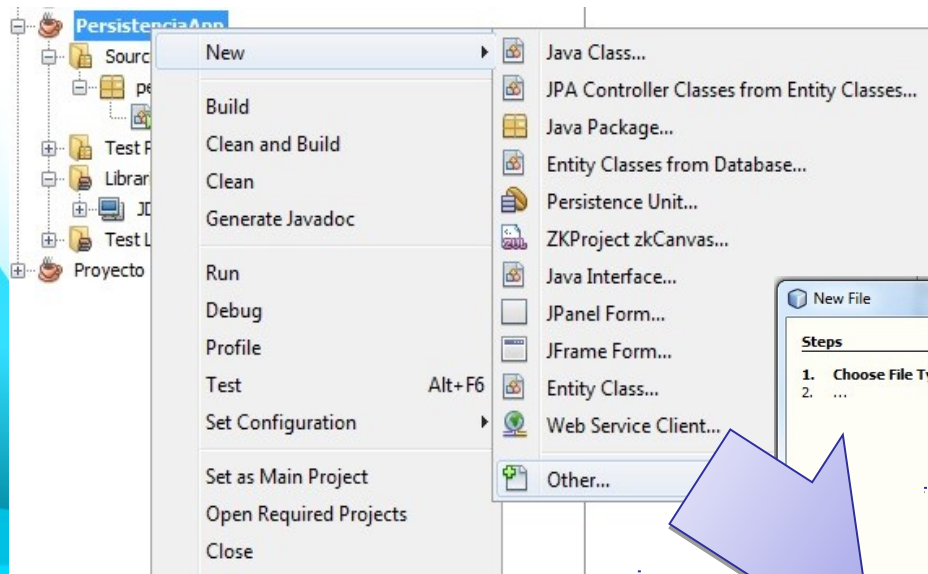


# Creando y configurando unidad de persistencia

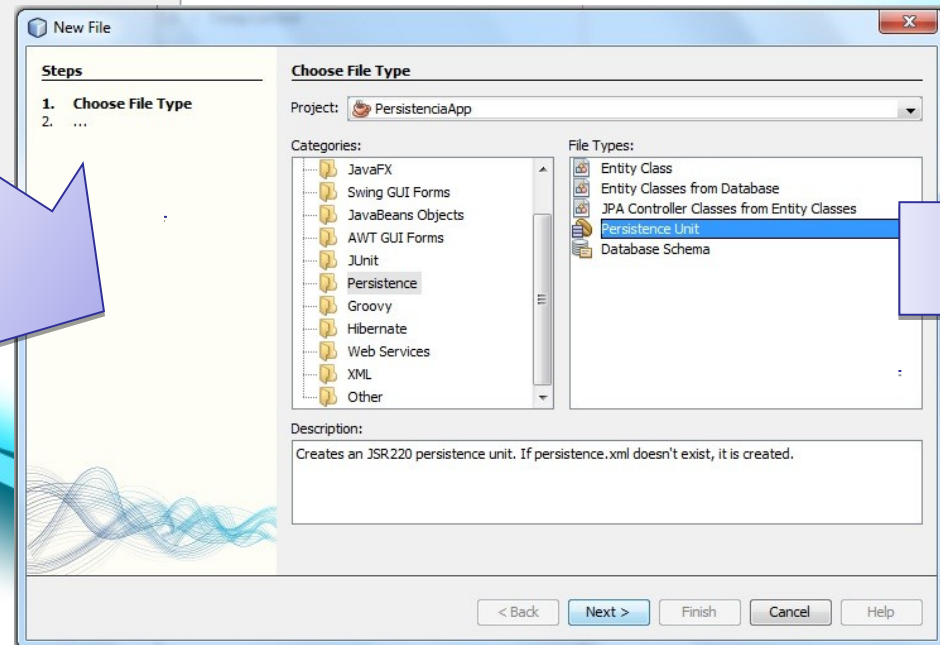


# Configurando Persistence Unit

Para iniciar la configuración de JPA en nuestro proyecto lo primero que debemos hacer es crear el archivo persistence.xml ,para hacer esto miremos como se hace :

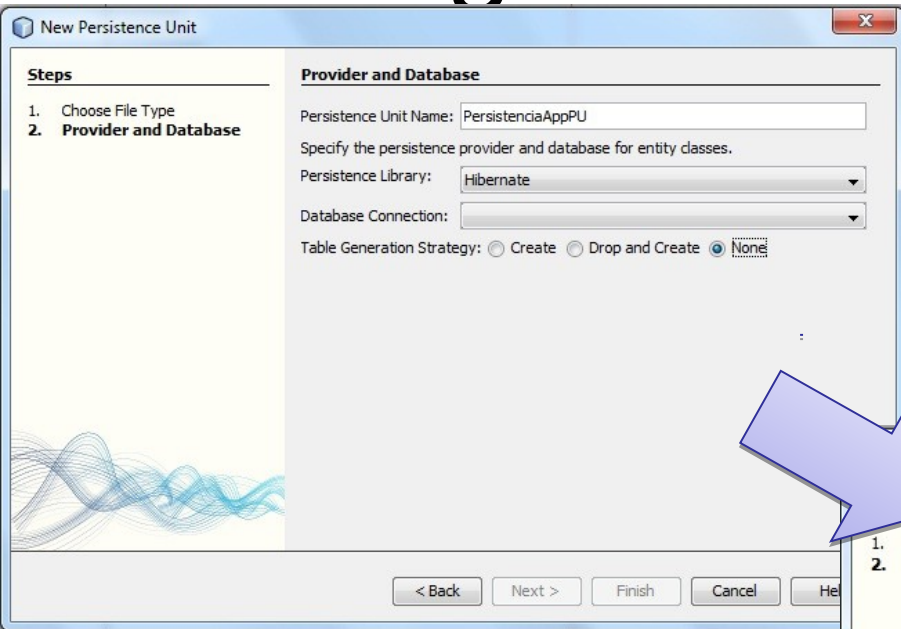


Click derecho sobre el proyecto,  
New -> Other ->persistence->Persistence Unit

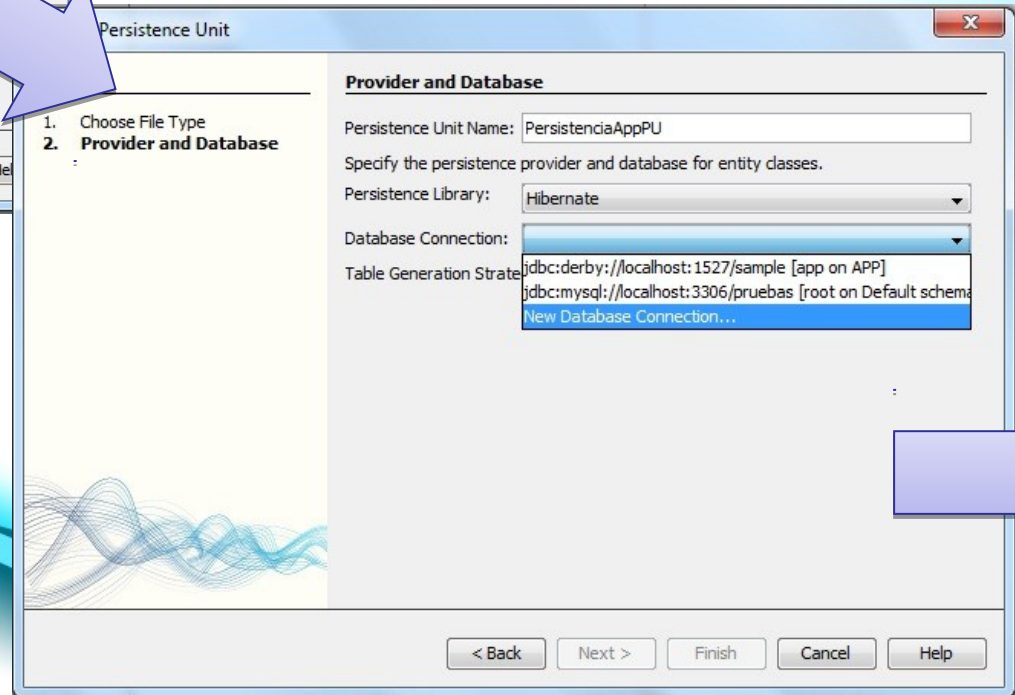


# Configurando Persistence unit

Seleccionamos el nombre a nuestra unidad de persistencia y también en este apartado indicamos con que ORM vamos a trabajar, Hibernate para nuestro ejemplo.

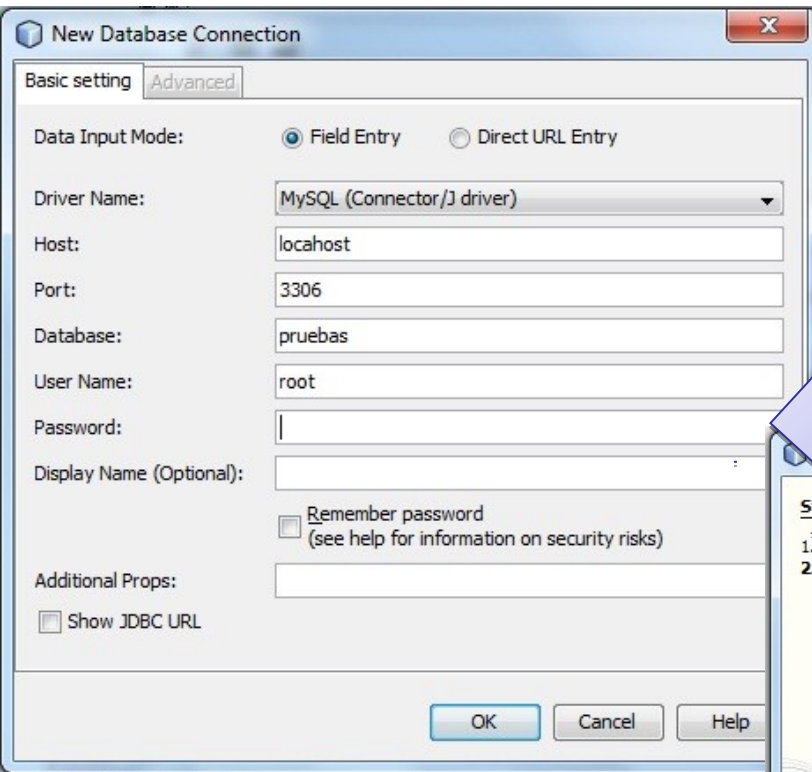


Configuramos nuestra conexión a nuestra base de datos



# Configurando cadena de conexión

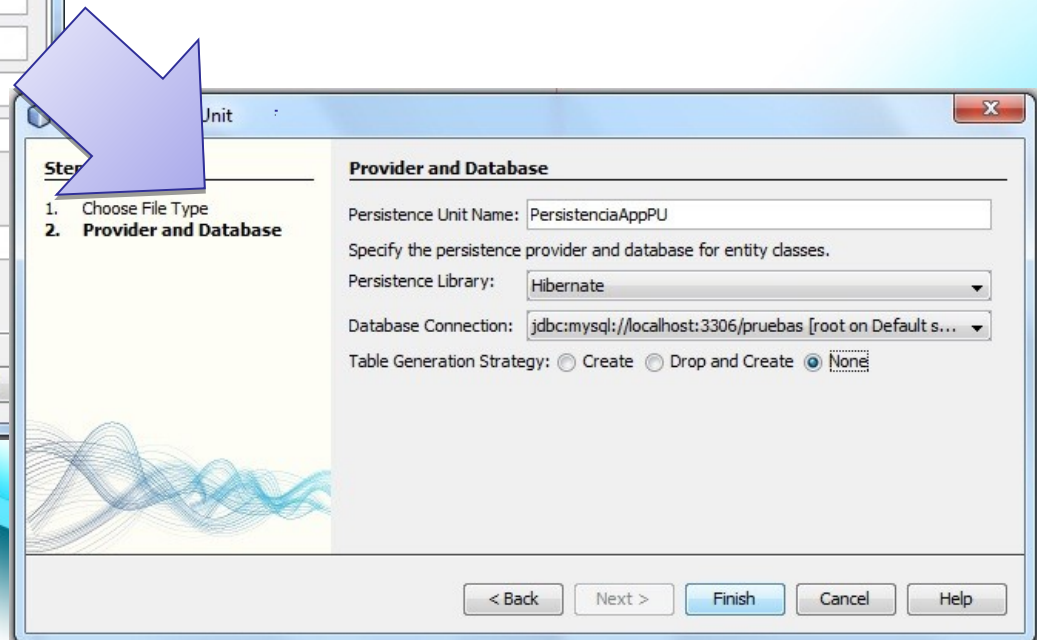
Establecemos todos los parámetros de conexión a nuestra base de datos.



The 'New Database Connection' dialog box is shown with the 'Basic setting' tab selected. It contains the following fields and options:

- Data Input Mode:** ☒ Field Entry, ☐ Direct URL Entry
- Driver Name:** MySQL (Connector/J driver)
- Host:** localhost
- Port:** 3306
- Database:** pruebas
- User Name:** root
- Password:** (empty field)
- Display Name (Optional):** (empty field)
- ☐ Remember password (see help for information on security risks)
- Additional Props:** (empty field)
- ☐ Show JDBC URL

Buttons at the bottom: OK, Cancel, Help.



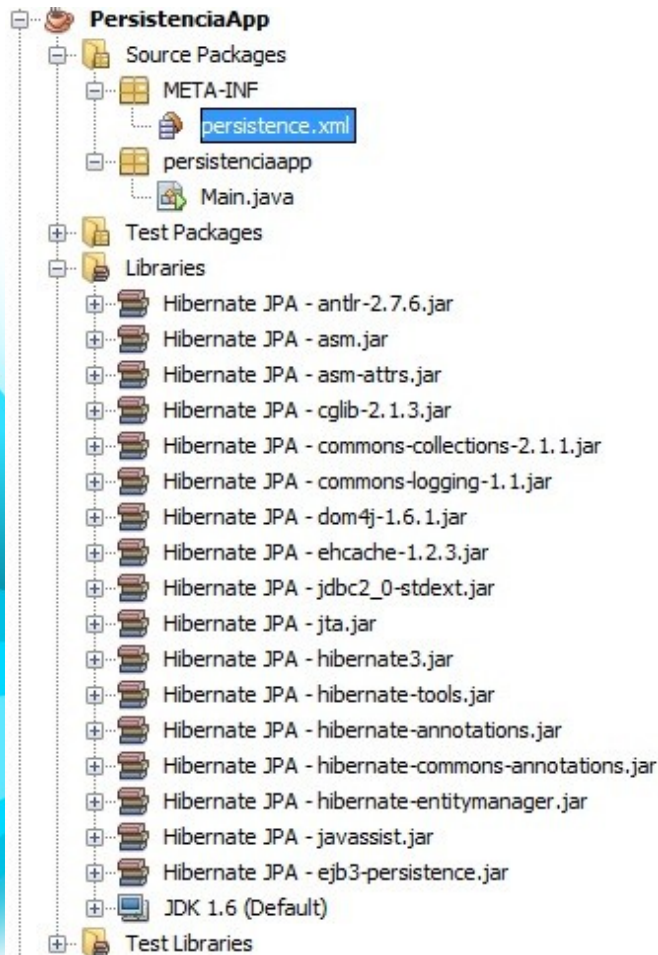
The 'Provider and Database' dialog box is shown, which is the second step in the configuration process. It contains the following fields and options:

- Persistence Unit Name:** PersistenciaAppPU
- Specify the persistence provider and database for entity classes.**
- Persistence Library:** Hibernate
- Database Connection:** jdbc:mysql://localhost:3306/pruebas [root on Default s...]
- Table Generation Strategy:** ☐ Create, ☐ Drop and Create, ☒ None

Buttons at the bottom: < Back, Next >, Finish, Cancel, Help.

Cuando este todo configurado le damos FINISH

# Configurando Persistence Unit



Luego de configurar la unidad de persistencia a través de Netbeans, nos agregara en las librerías todo lo necesario para usar JPA con Hibernate, además crea la Carpeta META-INF y le anexa el archivo de configuración persistence.xml.

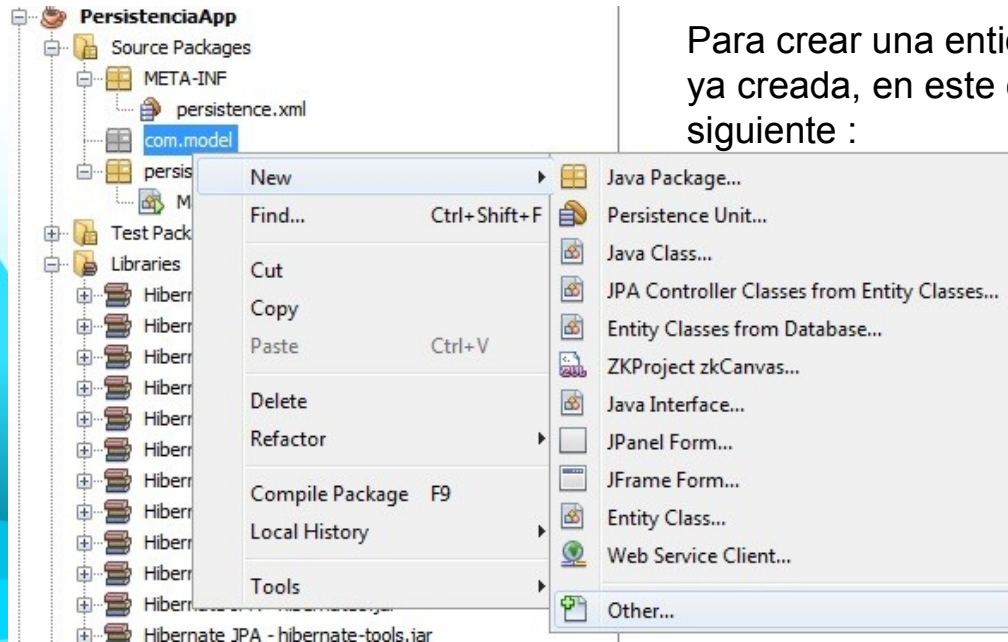


# Creando una entidad persistente



# Creando una entidad persistente

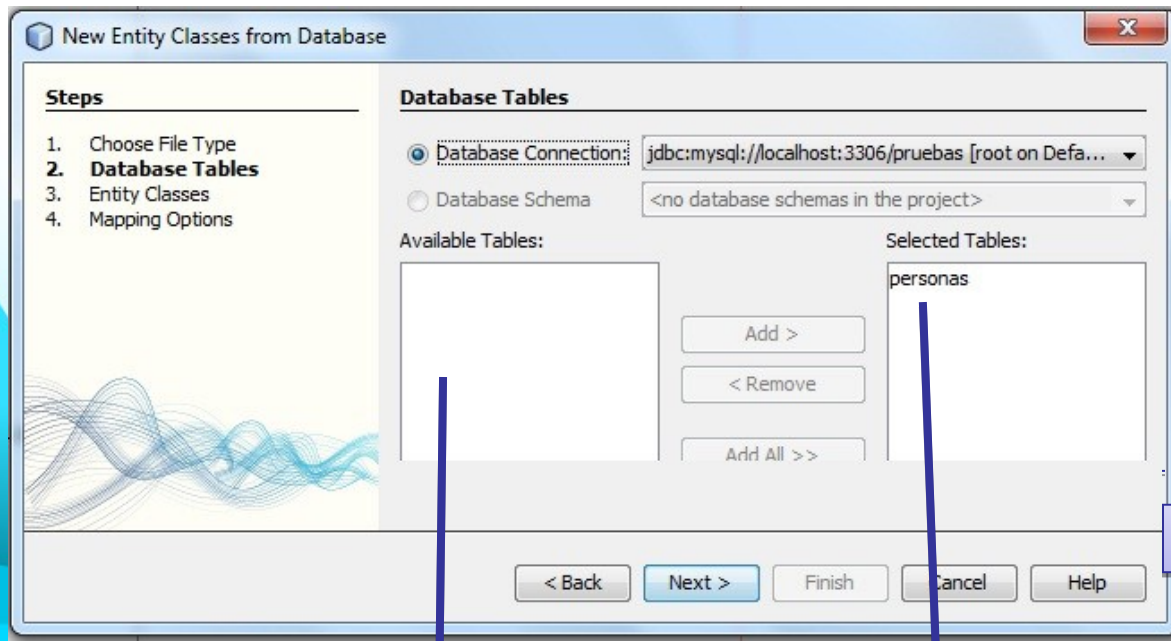
(Previamente he creado un paquete llamado **com.model** para agrupar mis clases de negocio.)



Para crear una entidad persistente lo vamos hacer de una tabla ya creada, en este caso ,la tabla Persona, para esto hacemos lo siguiente :

Click derecho sobre el paquete com.model,luego New -> Other ->persistence->**EntityClasses from Database**

# Creando una entidad persistente



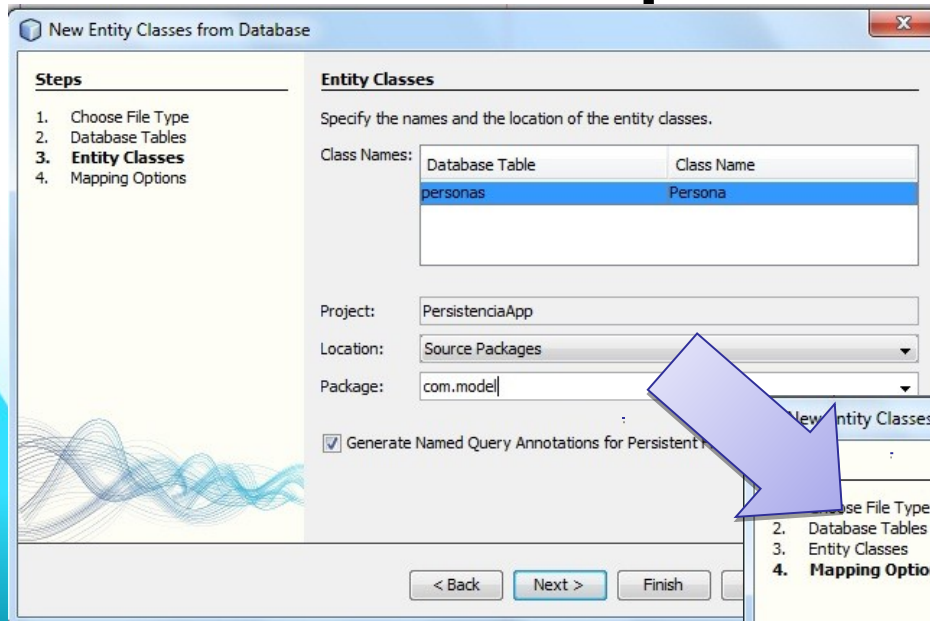
Seleccionamos nuestra cadena de conexión creada previamente.

Serán las tablas que se convertirán en clases persistentes. En este caso vamos a convertir la tabla **Personas** a una clase persistente llamada **Persona** con sus anotaciones.

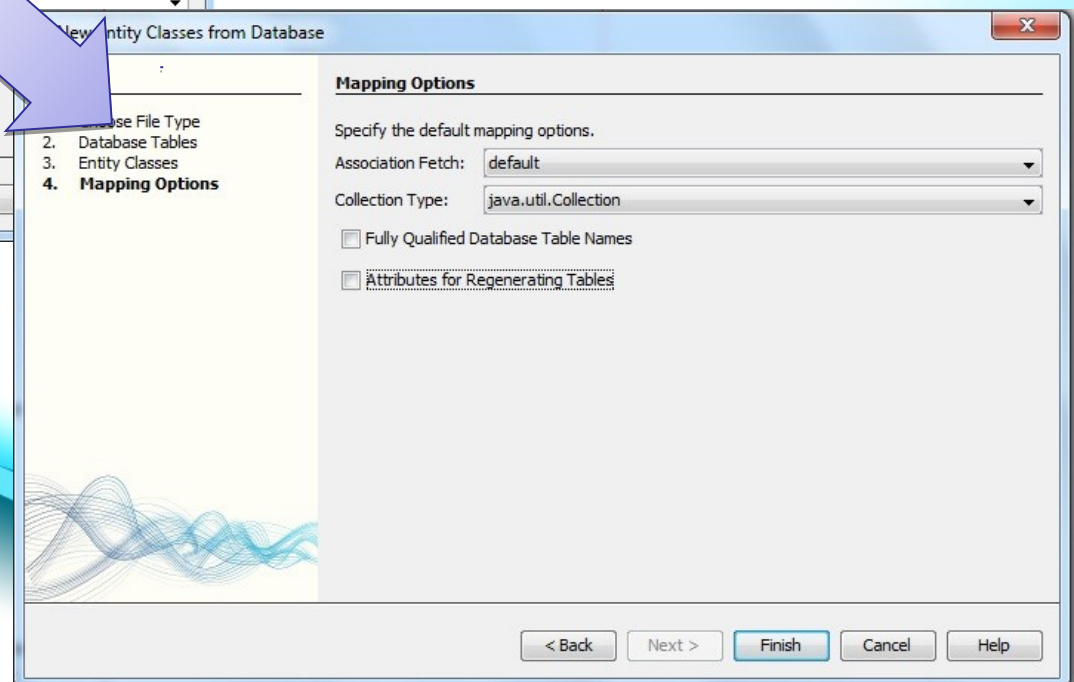
Son todas las tablas que nos arroja la conexión

# Creando una entidad persistente

Especificamos el nombre de nuestra entidad y su localización



Configuramos las diferentes opciones de mapeo como por ejemplo el tipo de asociaciones, con que interfaz se manejaran las colecciones, etc.



# Creando una entidad persistente

El resultado de este proceso es una clase con una serie de atributos anotados y propiedades :

```
package com.model;
```

```
import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
```

```
/* @author Ronald Cuello
 */
@Entity
@Table(name = "personas")
@NamedQueries({
    @NamedQuery(name = "Persona.findAll", query = "SELECT p FROM Persona p"),
    @NamedQuery(name = "Persona.findById", query = "SELECT p FROM Persona p WHERE p.id = :id"),
    @NamedQuery(name = "Persona.findByNombres", query = "SELECT p FROM Persona p WHERE p.nombres = :nombres"),
    @NamedQuery(name = "Persona.findByApellidos", query = "SELECT p FROM Persona p WHERE p.apellidos = :apellidos"),
    @NamedQuery(name = "Persona.findBySalario", query = "SELECT p FROM Persona p WHERE p.salario = :salario"),
    @NamedQuery(name = "Persona.findByEdad", query = "SELECT p FROM Persona p WHERE p.edad = :edad")})
public class Persona implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id")
    private Integer id;
    @Basic(optional = false)
    @Column(name = "nombres")
    private String nombres;
    @Basic(optional = false)
    @Column(name = "apellidos")
    private String apellidos;
    @Basic(optional = false)
    @Column(name = "salario")
    private double salario;
    @Basic(optional = false)
    @Column(name = "edad")
    private int edad;

    // SETS y GETS
```

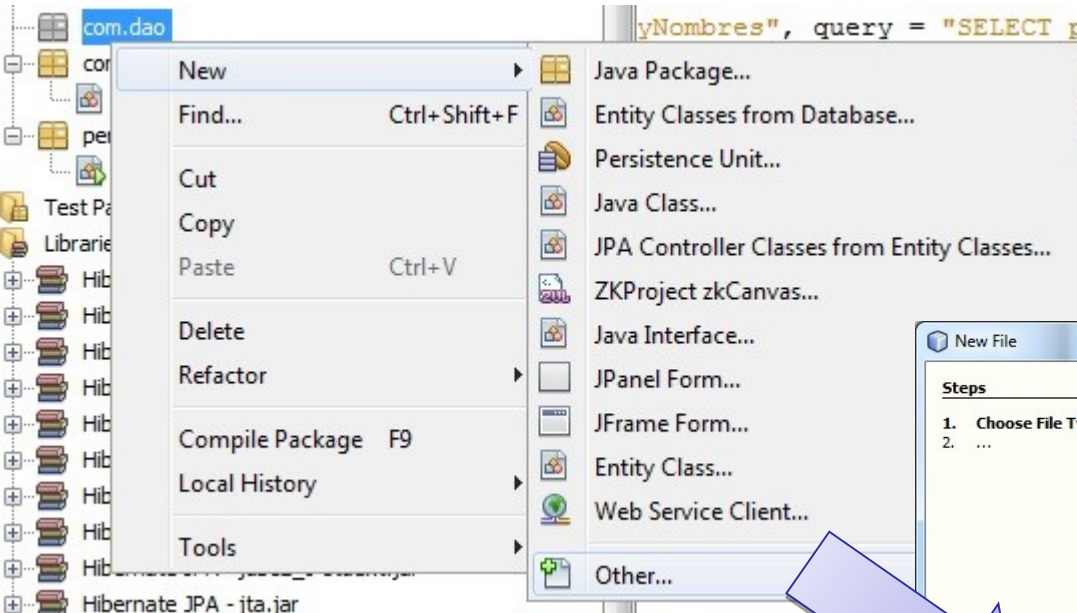


# Creando un JPA Controller



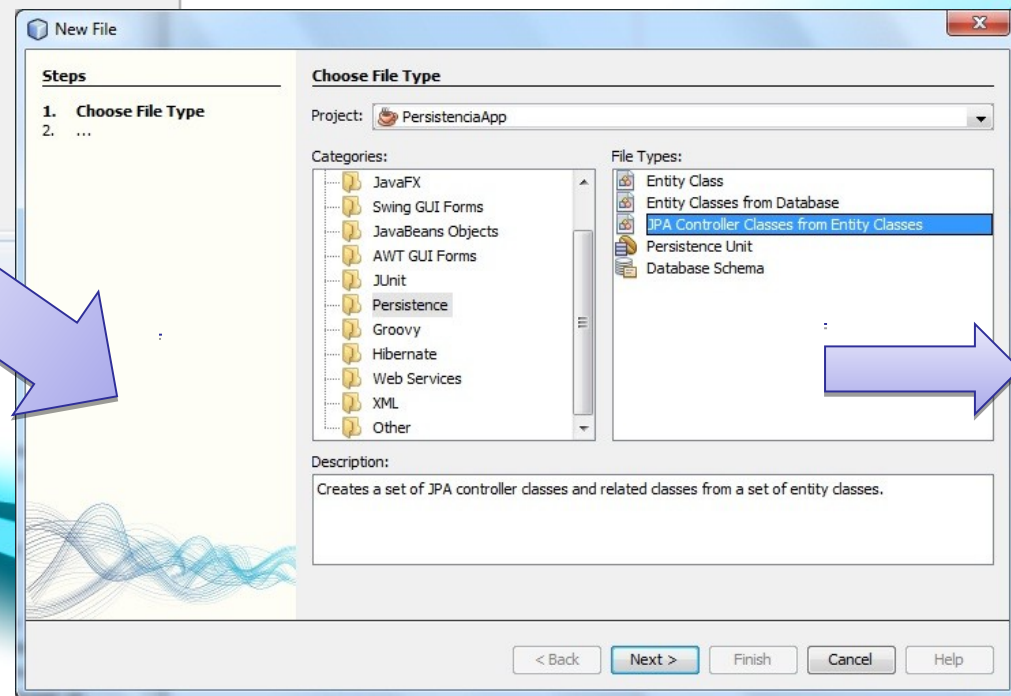
# Creando un JPA controller

(Previamente he creado un paquete llamado **com.dao** para agrupar mi capa de acceso a datos .)

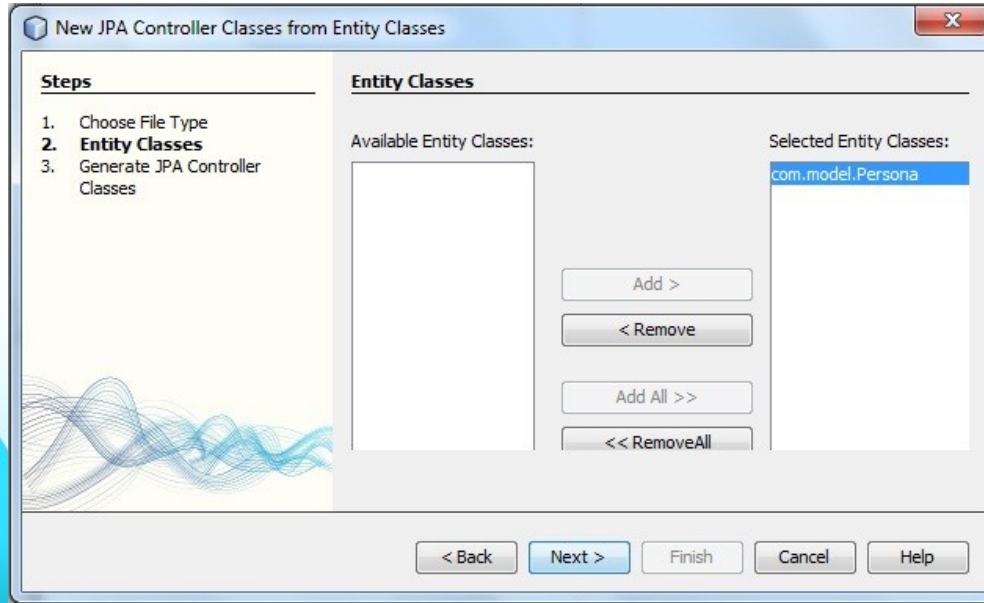


Para crear un controlador debemos hacer lo siguiente :

Click derecho sobre el paquete com.dao,luego  
New -> Other -> persistence-> **Jpa controller  
classe from Entity classes**



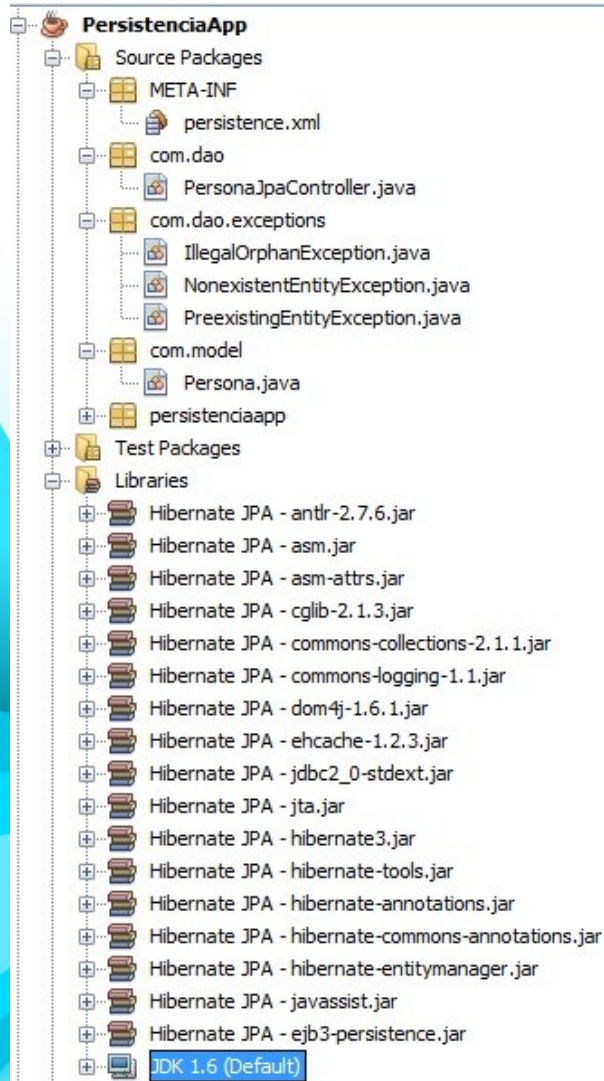
# Creando un JPA controller



Seleccionamos la entidad persistente a la que le vamos a crear su controlador de acceso a datos.

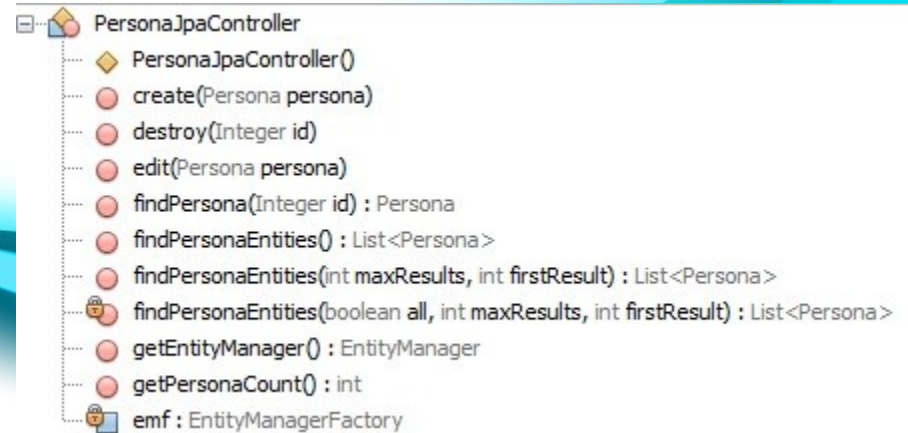
Le damos siguiente y Listo.

# Creando un JPA controller



Luego de terminar de configurar nuestro controlador ,Netbeans nos genera un paquete nuevo llamado com.dao.exceptions ,que serán las excepciones que lanzara nuestro controlador en caso de que ocurra un error.

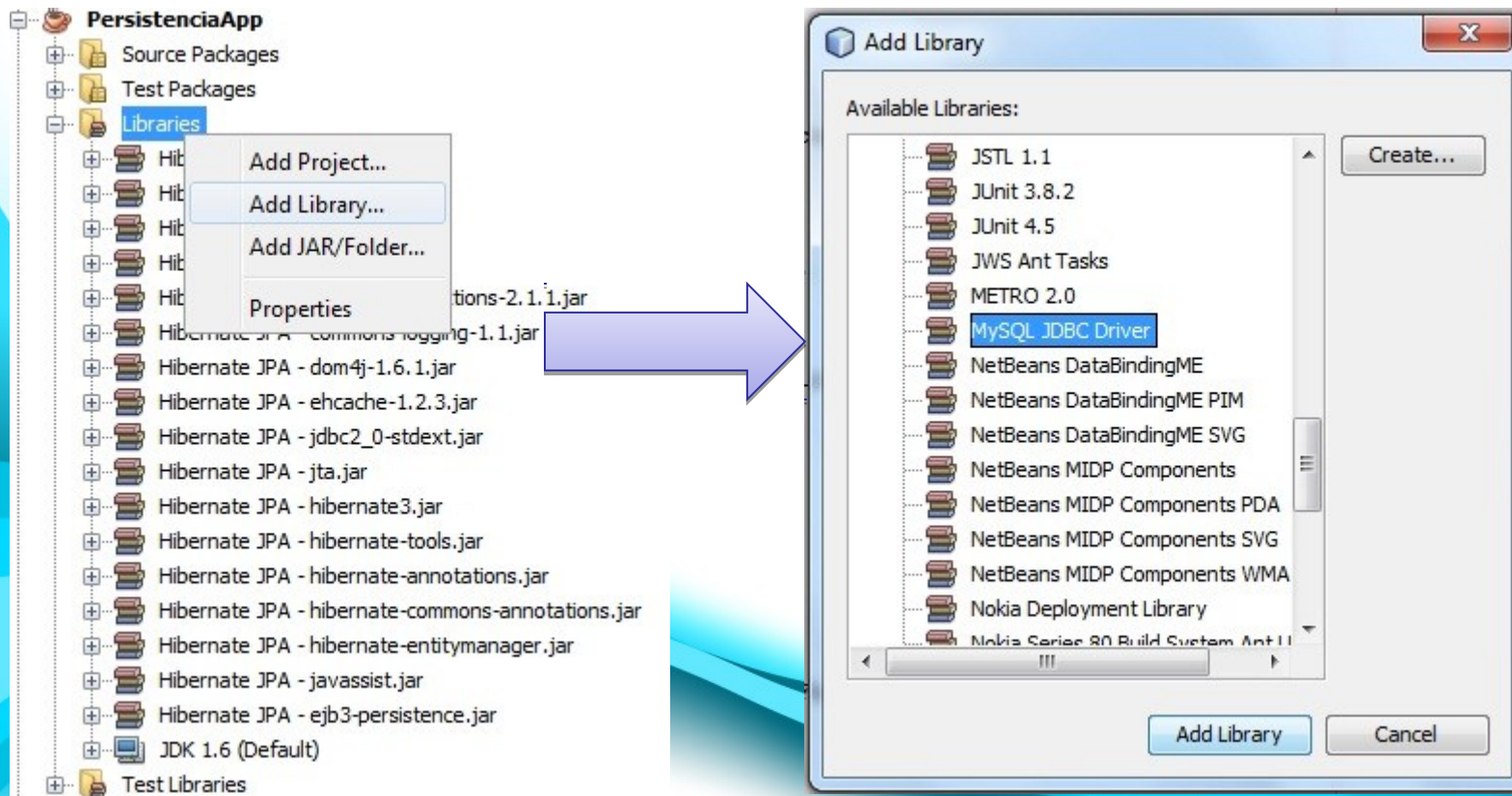
***(Métodos y atributos del controlador generado)***





# Agregar driver de conexión

Para terminar de configurar el proyecto faltaría por agregar el jar de conexión a MYSQL



# Veamos el resultado

```
package persistenciaapp;

import com.dao.PersonaJpaController;
import com.model.Persona;
import java.util.List;

/**
 *
 * @author Ronald Cuello
 */
public class Main {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        PersonaJpaController dao=new PersonaJpaController();
        List<Persona> lista=dao.findPersonaEntities();
        for(Persona persona : lista){
            System.out.println("Nombre "+persona.getNombres());
        }
    }
}
```



# En la vida real

No se si han dado cuenta, pero cuando netbeans nos genera un controlador, este controlador en su constructor inicializa el motor de persistencia, es decir, si creamos varias instancias de este controlador se inicializa varias veces el motor haciendo nuestra aplicación pesada y lenta.

Para resolver esto lo podemos hacer de varias formas : usando Spring, aplicando patrón singleton en los controladores o centralizando el EntityManagerFactory en una clase de utilerías.

Para este ejemplo usare una clase de utilería que me centralice el entityManagerFactory.

```
/**
 *
 * @author Ronald Cuello
 */
public class PersonaJpaController {

    public PersonaJpaController() {
        emf = Persistence.createEntityManagerFactory("PersistenciaAppPU");
    }
    private EntityManagerFactory emf = null;

    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }
}
```

# Creando clase de utileria

```
package com.util;

import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

/**
 *
 * @author Ronald Cuello
 */
public class JpaUtil {
    private static final EntityManagerFactory emf;
    static{
        try{
            emf=Persistence.createEntityManagerFactory("PersistenciaAppPU");
        }catch(Throwable t){
            System.out.println("Error a inicializar el Entity Manager Factory "+t);
            t.printStackTrace();
            throw new ExceptionInInitializerError();
        }
    }

    public static EntityManagerFactory getEntityManagerFactory(){
        return emf;
    }
}
```

# Código fuente

```
package com.util;

import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

/**
 *
 * @author Ronald Cuello
 */
public class JpaUtil {
    private static final EntityManagerFactory emf;
    static{
        try{
            emf=Persistence.createEntityManagerFactory("PersistenciaAppPU");
        }catch(Throwable t){
            System.out.println("Error a inicializar el Entity Manager Factory "+t);
            t.printStackTrace();
            throw new ExceptionInInitializerError();
        }
    }

    public static EntityManagerFactory getEntityManagerFactory(){
        return emf;
    }
}
```

# Modificando controlador

```
/**
 *
 * @author Ronald Cuello
 */
public class PersonaJpaController {

    public PersonaJpaController() {
        // emf = Persistence.createEntityManagerFactory("PersistenciaAppPU");
        emf=JpaUtil.getEntityManagerFactory();
    }
    private EntityManagerFactory emf = null;

    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }
}
```

Hacemos la corrección en el constructor por la clase que acabamos de crear

Listo eso es todo amigos.....

# Nos vemos

Para mas información sobre JPA visita mi blog

**[Ronaldcuello.blogspot.com](http://Ronaldcuello.blogspot.com)**

*Agradecer no cuesta nada ;)*