Proyecto Aplicación Java con Persistencia

<https://www.youtube.com/watch?v=KyAouX6UhRY>

<https://www.youtube.com/watch?v=l6kBC4Xz1pE>

<https://www.youtube.com/watch?v=HDNeHsLGBco>

<https://www.youtube.com/watch?v=lWeWrb6KkZQ>

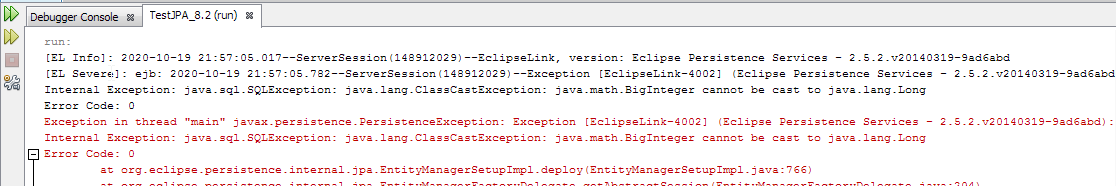
Creo un proyecto Java Ant del tipo Java Application.

Creo una conexión a BD de MySQL en el servidor de BD desde la solapa Server. con el conector-J.

En la carpeta Libaries del proyecto agrego el jar del connectorJ (tener muy en cuenta que versión se elige pues comencé utilizando

mysql-connector-java-5.1.23-bin.jar

y la aplicación me daba un error sobre el cual no podía encontrar información



con todos estos otros la aplicacion funcionó

mysql-connector-java-5.1.45-bin.jar

mysql-connector-java-5.1.47-bin.jar

mysql-connector-java-5.1.49-bin.jar

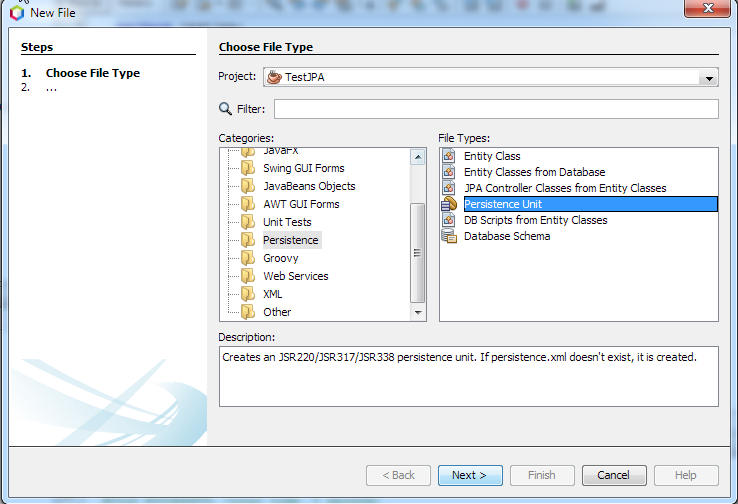
mysql-connector-java-8.0.19.jar

En NetBeans 11.3 solo funciona con el ConnectorJ 8.0.19.

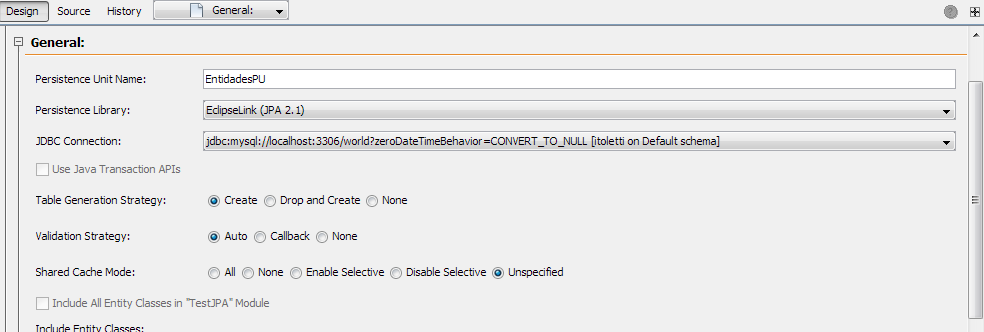
me di cuenta porque el primero me traía algunos inconvenientes para conéctame a algunas otras conexiones dentro de la solapa Service cuando utilizaba ese mismo conector.

Dentro de **SourcePackages** creo un paquete nuevo, para esto selecciono categoría Persistencia | Unidad de persistencia

lo hago del tipo de persistencia EclipseLink (JPA 2.1)



Esto nos creara el **META\_INF\persistence.xml** con todos los datos de la conexión



Aqui surge el concepto de anotaciones



Ahora vamos a ver como armamos el mapeo entre (clases) objetos y tablas nuestro proyecto en paquetes que representaran a cada una de las capas para el modelo de capas como funciona la persistencia de datos

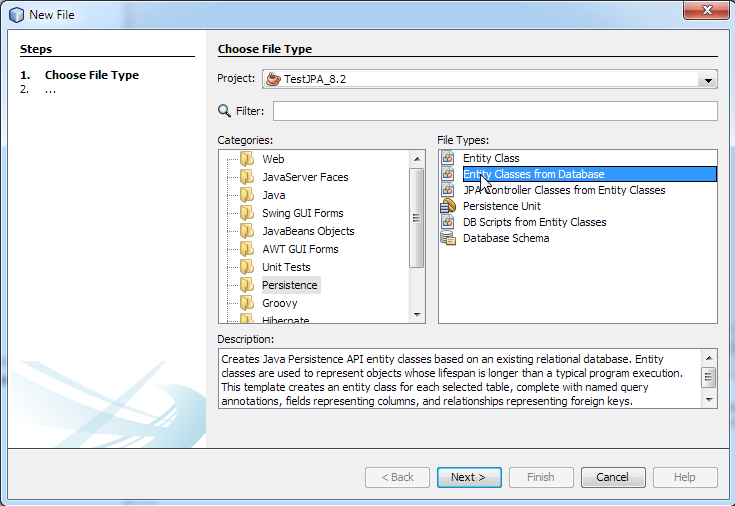
* un paquete para la lógica, contendrá las clases (Logica) (o Entidad)
* un paquete para intefáz grafica de los usuarios (GUI) (o Vista)
* un paquete para la persistencia (persistencia) (o Controlador)

esta separación, hace que cada capa sea independiente una de otra para cuando haga cambios, llegado el caso si tuviese que hacerlo.

La clase main() pertenece a la parte de la lógica por lo tanto lo podemos pasar al paquete Logica. Y vamos a crear allí las clases Country, City y CountryLenguage.

Estas podemos crearlas manualmente verificando las tablas de la BD y agregando las anotaciones correspondientes. Otra forma sería hacer automáticamente, es deci dejando que NetBeans lo haga por nosotros. Para esto ultimo hacemos new Persistence | Entity clases from Database indicándole el paquete donde creara las clases, la conexión a la BD de donde tomara las tablas, seleccionar las tablas para las cuales se desea crear entidades y algún otro parámetro mas.

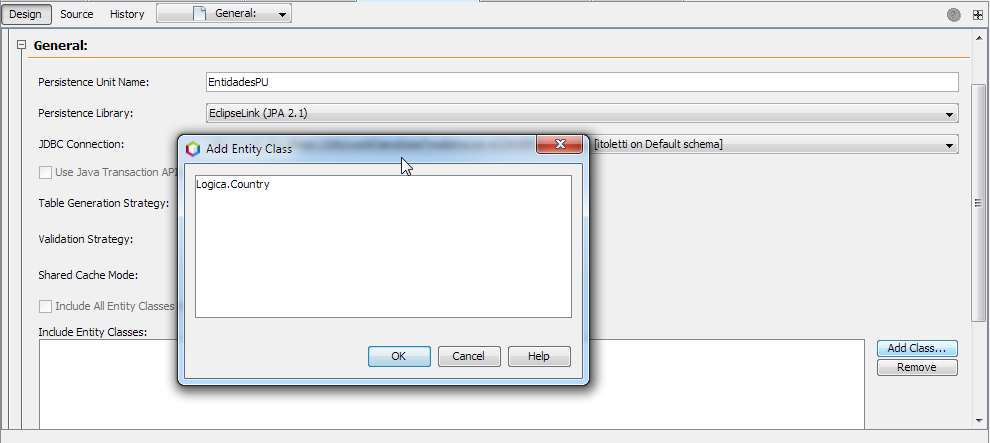
Este procedimiento además nos crea la Unidad de pesistencia (el paquete META\_INF y el archivo persistence.xml) y además adicionara los jar de JPA que requeire para funcionar en la carpeta Libraries



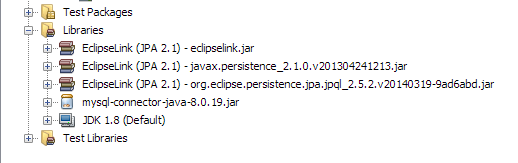
En un principio vamos a hacerel tratamiento manual para la primera entidad (la tabla Coutry). Dentro de paquete Logica vamos a crear una clase java Country. Creamos los atributos para Country del tipo prívate y con los tipos que se corresponan a los tipos de los mismos en la BD. Luego creamos un método constructor (vacio) y otro constructor con todos los atributos. Finalmente agregamos los metodos getters y setters y toString, para la entidad.

Con tinuamos agregando las anotaciones @Entity a la clase @id al atributo de la clase, @Basic a los demás (en el caso de fechas debiera de utilizar @Temporal(TemporalType.DATE) o @Temporal(TemporalType.Hour) que se corresponda con la clave de la tabla. Al principio me indicara error cuando agregue estas anotaciones, hasta que haga el import correspondiente de la clases javax.persistence. Entity, javax.persistence.Id, etc.

Una vez finalizada la definición de la clase de persistencia debemos de agregarla a nuestro archivo MATA\_INF\persistencia.xml

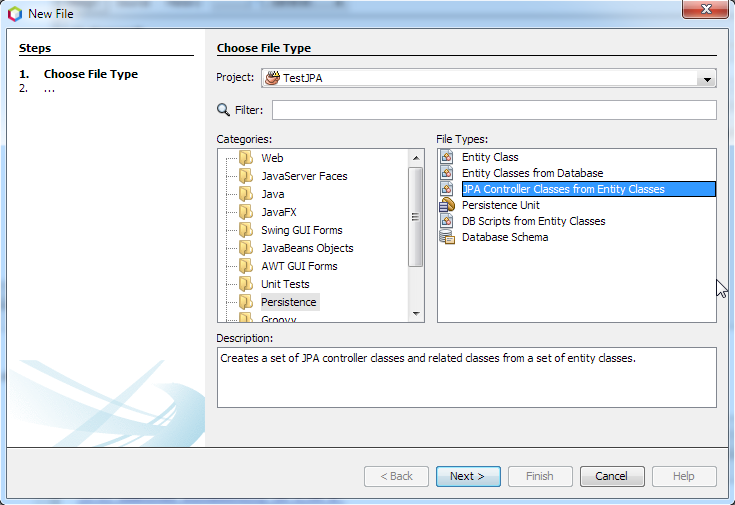


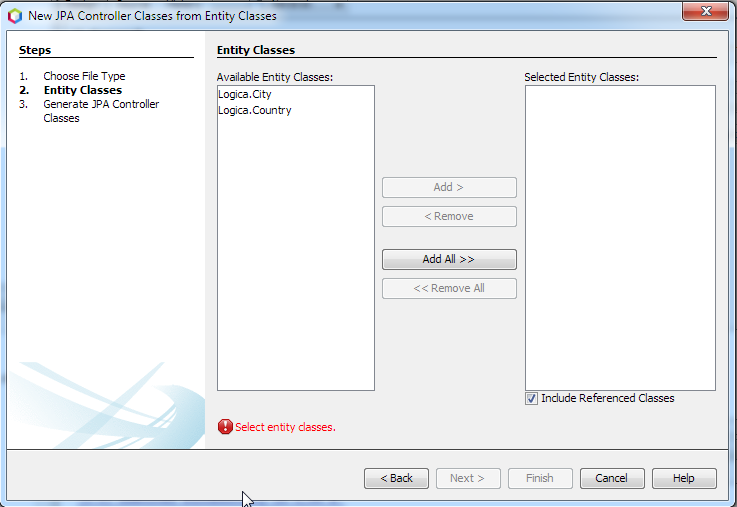
Ademass debemos agrega a mano las librerías de JPA



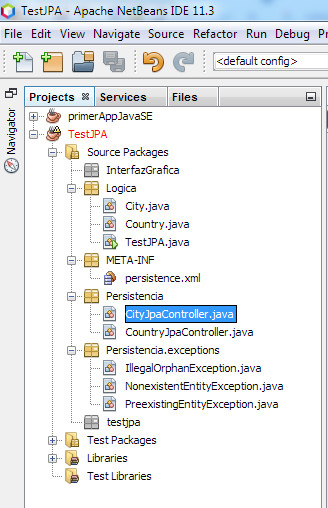
Esto ultimo ya mencione que se hace de manera automática creando las entidades mediante new Persistence|Entity clases frm Database.

El siguiente paso será generar los controladores, definimos las unidades de persistencia a hora en el paquete persistencia vamos a crear los controladores que relacionarán las clases de entidades con sus correspondientes objetos en la BD, para esto dentro del paquete Persistencia vamos a crear clases de controladores para las clases de entidades seleccionamos entonces Persistencia| JPA Class Controller from Entity Class

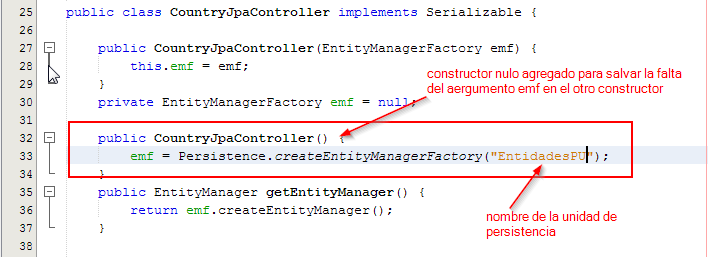




Esto nos crea una clase en el paquete persistencia una clase JPAController para cada clase de entidad esta tendrá los métodos para crear, modificar, eliminar y buscar una entidad determinada. Además el proceso crea también un paquete Persistencia.exception con clases para manejar las posibles excepciones que se puedan generar.



Dentro de las clases **JPAController** que nos crea para City y para Country vamos a ver que además de los métodos mencionados dispone un método constructor con un argumento del tipo **EntityManagerFactory**, pero no tenemos ninguna **EntityManagerFactory** creada para poder pasarle. Entonces vamos hacerlo de otra manera, vamos a permitirle q JPA se encargue de crear esta **EntityManagerFactoy** a partir de otra que estamos utilizando en la misma clase que estamos definiendo. En el código que nos generó vemos que hay un atributo emf (del tipo **EntityManagerFactory**) definido en esta clase e inicializado a null. Entonces vamos a crear otro constructor CountryJPAController, para la clase (además del que se creo automáticamente) en este caso el constructor vacio, sin argumentos, pero que utiliza en su código el atributo emf definido.

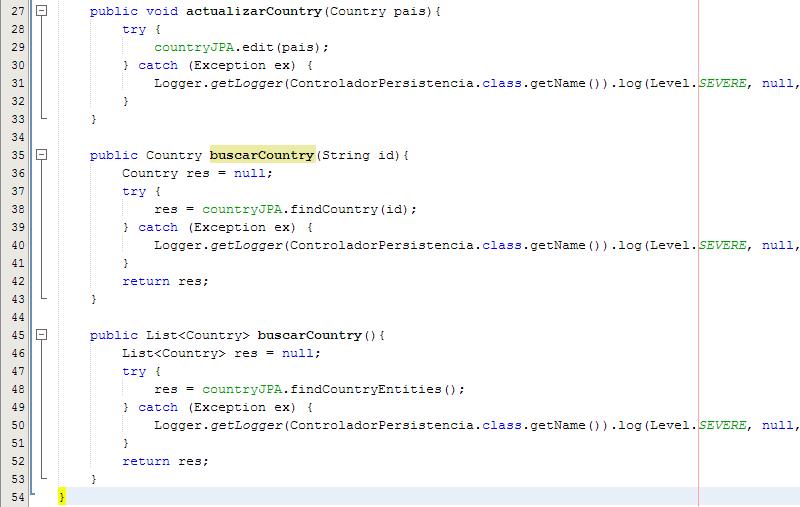


Ahora necesitamos crear una clase controladora de persistencia… ¿para que esto?... para poder llamar desde la lógica a los controladores que en este caso son de JPA, pero se podría haber elegido trabajar con algún framework de persistencia. En algunos otros caso a esto lo he visto mencionado como el Servicio (en particular en Javascrip, Typescript con Nest)

Entonces creo una clase Controlador de persistencia con miembros de los objetos controladores de las diferentes clases de entidades a persistir. Creo una instancia de cada uno para luego desde el código poder referenciarlos a través de sus nombres

Si se quiere esto es el único código que escribimos ya que la ded}finicion de las entidades de persistencia asi como sus controladores las puede crear auntomaticamente NetBeans





Desde el método main podemos llamar a los métodos de esta clase para ver funcionar el CRUD a manera deconsola, también se puede con SWING hacer un formulario estilo ABM para hacer una aplicación de escritorio de tipo Visual. Y si se desea hacer un API RESTful

Este hace lo mismo con una aplicación Java escritorio pero con formularios (Swing)

Conexion Java Mysql con JPA (CRUD) Configuración del Proyecto

<https://www.youtube.com/watch?v=ESAUuQfdnRY>

Conexion Java Mysql con JPA (CRUD) Leer Datos

<https://www.youtube.com/watch?v=jmbrEBj7HuI>

Conexion Java Mysql con JPA (CRUD) Crear Registros

<https://www.youtube.com/watch?v=fz2qiKARF-c>

Conexion Java Mysql con JPA (CRUD) Eliminar Registros

<https://www.youtube.com/watch?v=L9FHQtHYTl8>

Conexion Java Mysql con JPA (CRUD) Actualizar Registros

<https://www.youtube.com/watch?v=hmUFBfMB8wI>

# **Utilizando JPA NamedQueries**

<https://www.arquitecturajava.com/utilizando-jpa-namedqueries/>

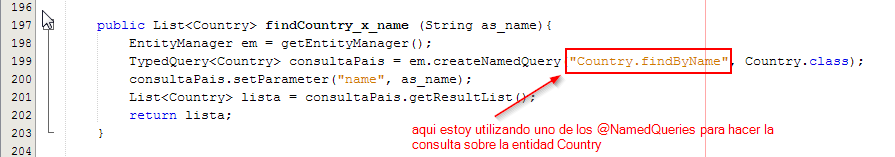
Cuando creamos la clase Entidad a apartir de la BD en el archivo de la clase se ven este tipo de anotaciones @NamedQuery



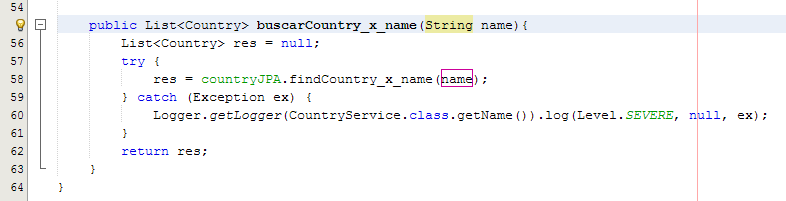
Just antes de arrancar con la definición de la clase.

Estas se pueden utilizar utilizando un objeto EntityManager creado desde EntityManagerFactory.

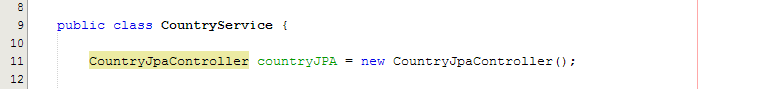
Por ejemplo en el controlador CountryController además de los métodos que me crea automáticamente NetBeans puedo agregar uno propio con el fin por ejemplo de buscar los países por el nombre del país… como no es clave existe la posibilidad de que me retorne más de uno.



Luego en lo que di a llamar el servicio CountryService



donde **countryJPA** es una instancia del **CountryController** en el **ServiceController**



Finalmente en alguna parte del código en el main por ejemplo usamos

