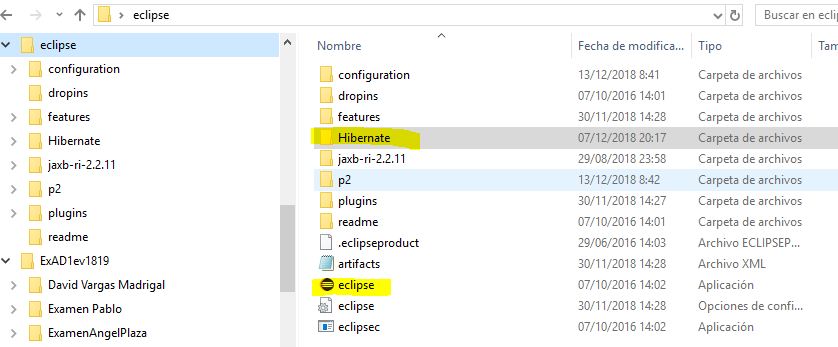
PRACTICE 3.2. PROGRAMMING WITH HIBERNATE IN ECLIPSE

OBJECTIVE: Create a program to access a database and work with it using Hibernate

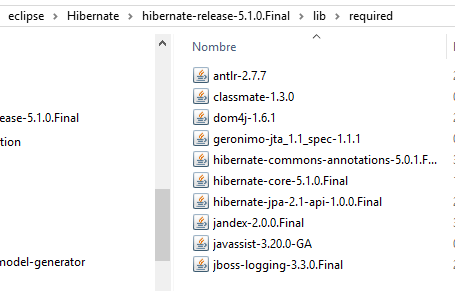
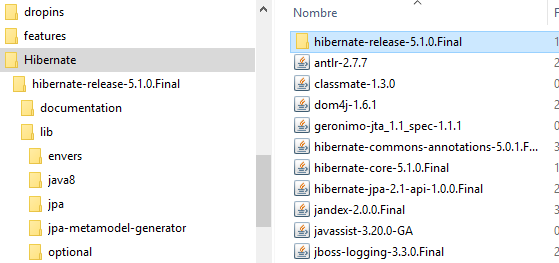
# PRACTICE:

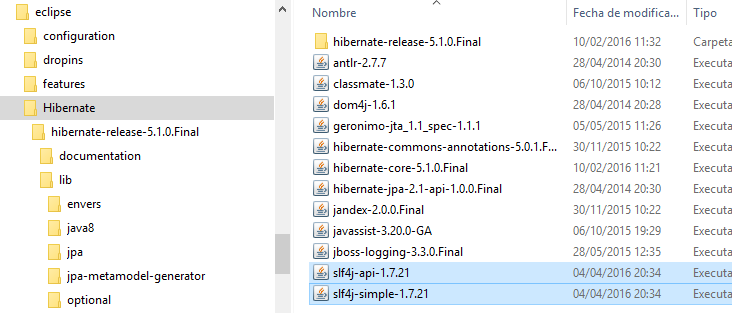
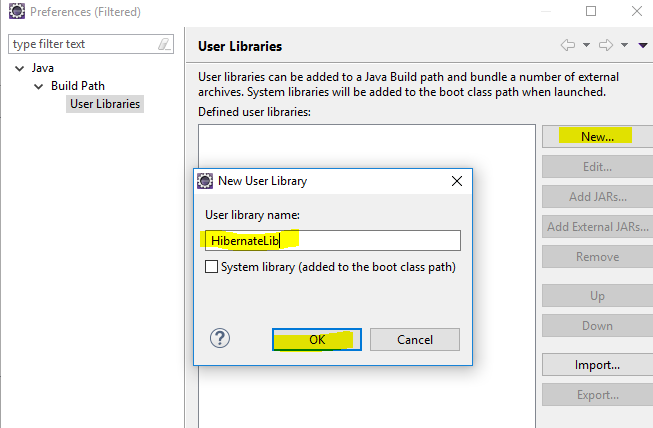
STEP 1:Install Hibernate in Eclipse: (in practices 3-1 we installed “Hibernate tools” to create the configuration files). Now we need the libraries to use Hibernate Objects (session, transactions, etc.)

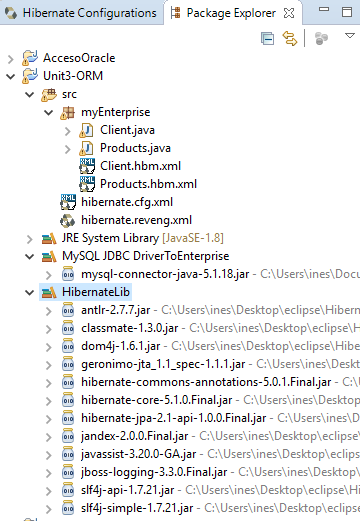
* download **hibernate-release-5.1.0.Final.zip**
* Create a new Folder in you eclipse neon folder. Unzip **hibernate-release-5.1.0.Final.zip inside Hibernate folder.**



* From the directory …\**eclipse\Hibernate\hibernate-release-5.1.0.Final\lib\required**, select all the files and copy them in eclipse\Hibernate

 🡺

* Download the libray slf4j fro [www.slf4j.org](http://www.slf4j.org), the file slf4j-1.7.21.zip. Unzip the folder. Find the files slf4j-api-1.7.21.jar and slf4j-simple-1.7.21.jar and copy them in eclipse/Hibernate.
* 
* In our project, Select Build Path>>Add Libraries>> User Library>>Next
* 
* Then “Add External JARs…” and select all files in eclipse/Hibernate. Press Finish



STEP 2: CREATE CLASS HibernateUtil.java

The class will open a sessionFactory for all other classes that need connection to the database

**(Note: this step is optional. We could use the class org.hibernate.SessionFactory and load a value with**

**SessionFactory** sf=new Configuration().configure().buildSessionFactory)

**With this class HibernateUtil we will use it later with the code:**

**SessionFactory** *sf* = HibernateUtil.*getSessionFactory*();

**package** myEnterprise;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.boot.registry.StandardServiceRegistryBuilder;

**import** org.hibernate.cfg.Configuration;

**public** **class** HibernateUtil {

**private** **static** **final** SessionFactory ***sessionFactory***=*buildSessionFactory*();

**private** **static** SessionFactory buildSessionFactory(){

**try**{

**return** **new** Configuration().configure().buildSessionFactory(**new** StandardServiceRegistryBuilder().configure().build());

}**catch**(Throwable e){

System.***out***.println("Initical SessionFactory creation failes."+e);

**throw** **new** ExceptionInInitializerError(e);

}

}

**public** **static** SessionFactory getSessionFactory(){

**return** ***sessionFactory***;

}

}

STEP 3: CREATE OUR APPLICATION. It will access the database Enterprise and will have a method to insert products in the table PRODUCTS.

We will use the classes: SessionFactory, Session, Transaction

First, let’s create the class structure. We will add methods later.

Structure:

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.Configuration;

**import** myEnterprise.\*;

**public** **class** HibernateEnterprise {

**private** **static** SessionFactory *sf*; //this SessionFactory will be created once and used for all the connections

HibernateEnterprise(){//constructor

*sf* = HibernateUtil.*getSessionFactory*();

//sf=new Configuration().configure().buildSessionFactory //also works

}

**public** **void** addProduct(**int** id,String name, **double** price ){

Session session=*sf*.openSession();

Transaction tx=**null**;

//create the product with the parameters in the method

Products p=**new** Products();

p.setName(name);

p.setPrice(price);

p.setProductId(id);

//keep it in the database=🡺session.save(p)

**try**{

System.***out***.printf("Inserting a row in the database: %s, %s, %s \n",id,name,price);

tx=session.beginTransaction();

session.save(p);//we INSERT p into the table PRODUCTS

tx.commit();//if session.save doesn't produce an exception, we "commit" the transaction

}**catch**(Exception e){//if there is any exception, we "rollback" and close safely

**if** (tx!=**null**) {

tx.rollback();

}

}**finally**{

session.close();

}

}

}//class

To try our code, write the class UseHibernateEnterprise:

**public** **class** UseHibernateEnterprise {

**public** **static** **void** main(String[] args) {

HibernateEnterprise h=**new** HibernateEnterprise();

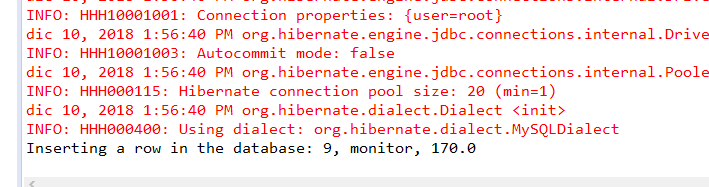
h.addProduct(9, "monitor",170);

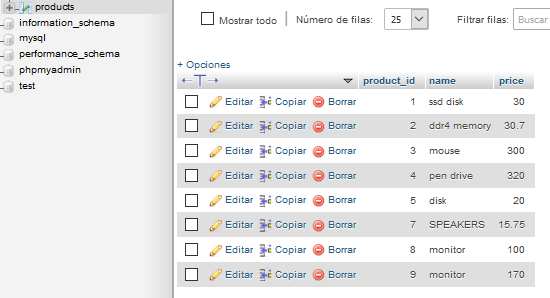
}

}

IMPORTANT: when we execute the code, it seems that we have errors, but it is working. What happens is that the session logger shows the INFO about the connection. I have tried to change the level of the logger to ERROR only, but I can’t do it. If you discover how to turn OFF the logger INFO, please tell me.

We get INFO messages, but it is executed ok:





STEP 4: Write all the methods that your application needs to work with the database. We will do a method to find a product by the id. To locate the product, we will use the session method load:

p=(Products)session.load(Products.**class**, identifier\_of\_Products);

**public** Products findProductById(**int** id){

Session session=*sf*.openSession();

Transaction tx=**null**;

Products p=**new** Products();

**try**{

System.***out***.println("loading the object from the database");

tx=session.beginTransaction();

p=(Products)session.load(Products.**class**, id);

tx.commit();

System.***out***.println("The product with id="+id+" is:"+p.getName());

}**catch**(ObjectNotFoundException e){

**if**(tx!=**null**){

System.***out***.println(e);

System.***out***.println("Product not found");

}

}**catch**(Exception e){

**if**(tx!=**null**){

System.***out***.println(e);

tx.rollback();

}

}

**finally**{

session.close();

}

**return** p;

}

Try the code and run it:

**public** **class** UseHibernateEnterprise {

**public** **static** **void** main(String[] args) {

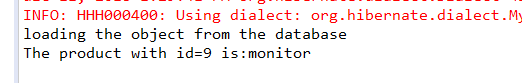
HibernateEnterprise h=**new** HibernateEnterprise();

//h.addProduct(9, "monitor",170);

h.findProductById(9);

}

}



STEP 5: DELETE A PRODUCT KNOWING THE ID:

**public** **void** deleteProductById(**int** id){

Products p=**new** Products();

Session session=*sf*.openSession();

Transaction tx=**null**;

**try**{

tx=session.beginTransaction();

p=(Products)session.load(Products.**class**, id);

session.delete(p);

tx.commit();

System.***out***.printf("Object deleted FROM THE DATABASE: %s, %s, %s \n",p.getProductId(),p.getName(),p.getPrice());

}**catch**(Exception e){

**if**(tx!=**null**){

tx.rollback();

}

}**finally**{

session.close();

}

}

DO BY YOURSELF:

1)Try it to see if it works:

2) What happens if you try to delete an id which doesn’t appear in the database? Try to p.delete(25) and see what happens.

3) Change your code so that if the product doesn’t exist, the code won’t abort, but will show you a message “Product not found with id:25. Deleting cannot be completed”.

STEP 5: UPDATE A PRODUCT IN THE TABLE

**public** **void** updateProductById(**int** id,String newName,**double** newPrice){

Products p=**new** Products();

Session session=*sf*.openSession();

Transaction tx=**null**;

**try**{

tx=session.beginTransaction();

System.***out***.println("Updating a value");

System.***out***.println("Before updating, we need to load the object");

p=(Products)session.load(Products.**class**, id);//we load the pen drive

p.setPrice(newPrice);//we change the properties

p.setName(newName);

session.update(p);//we update the values in the database

tx.commit();

System.***out***.println("Object updated");

}**catch**(Exception e){

System.***out***.println(e);

**if**(tx!=**null**){

tx.rollback();

}

}**finally**{

session.close();

}

}

QUESTIONS:

1. Does it fail if the product doesn’t exist? If it fails, made changes to the code to solve it.
2. In the class UseHibernateEnterprise, try to update 2 products.