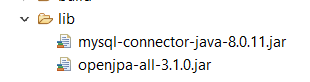
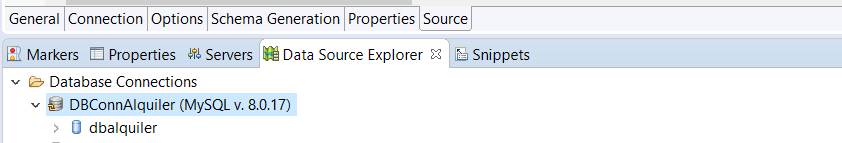
**Relación de Herencia**

Crear un proyecto JPA (JPATest2)

Crear una carpeta lib donde copiaremos los jar de mysql y openjpa. Agregar los dos jar al buildpath

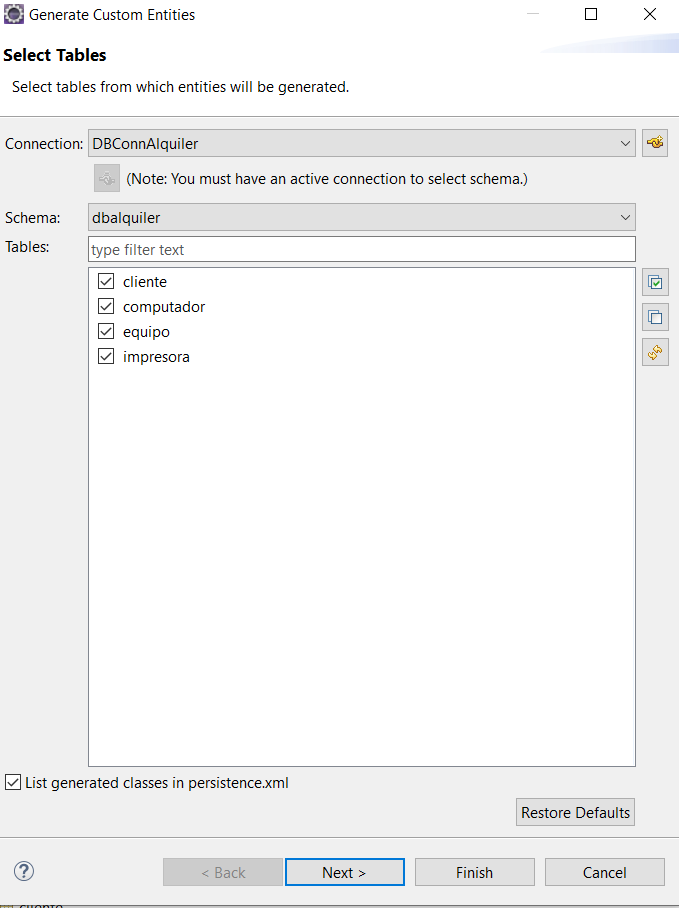


Nos conectamos a la base de datos con la que vamos a trabajar

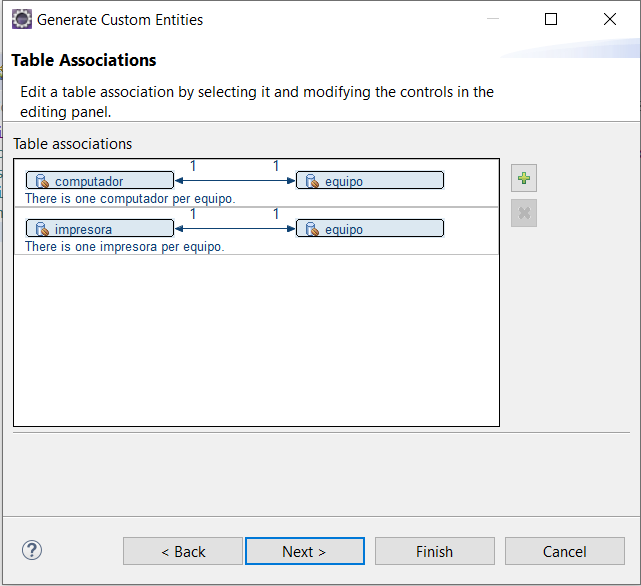


Sobre el proyecto JPATest2, creamos un nuevo paquete app.modelo. En este paquete damos clic en New -> JPA Entities from Tables

Al seleccionar la conexion vemos las tablas presentes



Seleccionamos y damos click en Next. En la siguiente ventana vemos las relaciones entre las tablas. Aquí se muestra la relación de herencia que hemos modelado

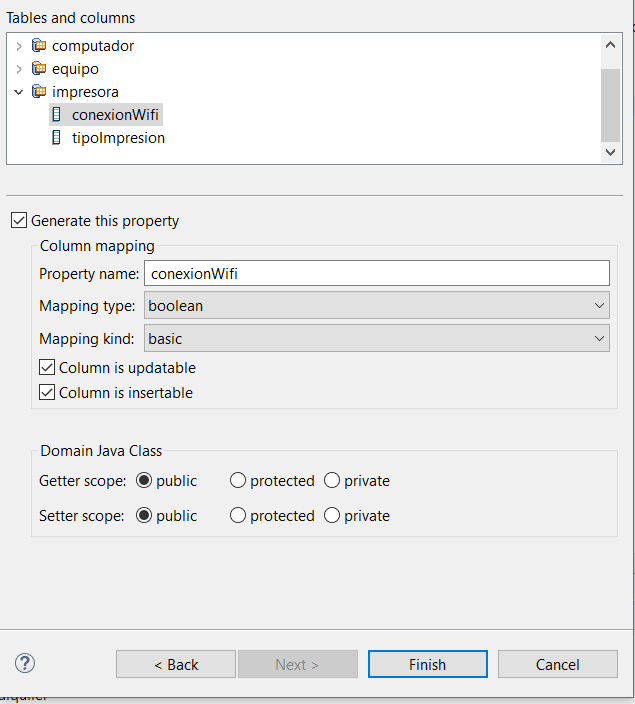


Damos click en Next y en la siguiente ventana dejamos los valores marcados y Next

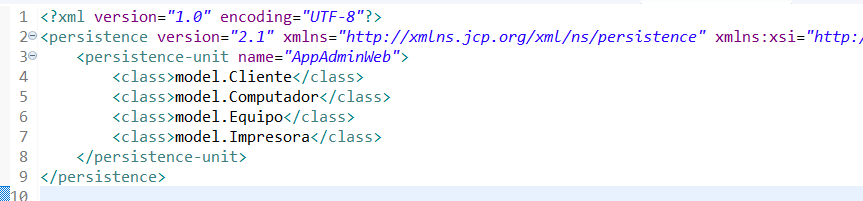
En la siguiente ventana customizamos los campos de las tablas

Cliente, equipo y computador sin cambios

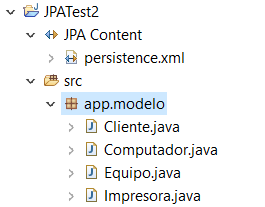
En la tabla impresora cambiaremos el tinyint a boolean y dar click en Finish



Las 4 clases fueron agregadas al archivo xml



Y el src del proyecto vemos que se agregaron los 4 archivos .java



Agregar en el archivo xml la informacion de la conexion

<properties>

<property name="openjpa.RuntimeUnenhancedClasses" value="supported"/>

<property name="openjpa.ConnectionURL" value="jdbc:mysql://localhost:3306/dbalquiler"/>

<property name="openjpa.ConnectionDriverName" value="com.mysql.jdbc.Driver"/>

<property name="openjpa.ConnectionUserName" value="root"/>

<property name="openjpa.ConnectionPassword" value="gladys123"/>

<property name="openjpa.Log" value="DefaultLevel=WARN, Tool=INFO"/>

</properties>

Ahora cambiaremos las clases Equipo, Computador e Impresora para reflejar la relación de herencia

En la clase Equipo

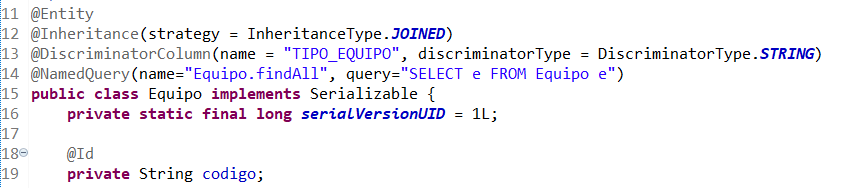
La estrategia que estamos utilizando es Joined, hay que indicar en la clase

Luego de la notacion @Entity podemos escribir

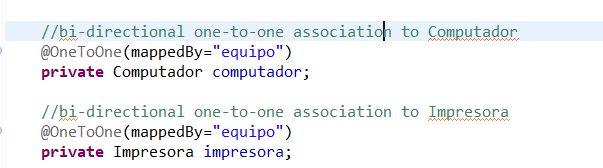
@Inheritance(strategy = InheritanceType.JOINED)

La tabla equipo tiene un campo que va a servir para diferenciar si es computadora o impresora, ese campo hay que agregar como discriminador

@DiscriminatorColumn(name = "TIPO\_EQUIPO", discriminatorType = DiscriminatorType.STRING)



Y eliminamos los atributos de la relación 1 a 1 con computador e impresora y también los setters & getters relacionados



Agregamos un constructor parametrizado

public Equipo(String codigo, String description, String marca, float costoAlquiler) {

this.codigo = codigo;

this.description = description;

this.marca = marca;

this.costoAlquiler = costoAlquiler;

}

y el toString

@Override

public String toString() {

return " [codigo=" + codigo + ", costoAlquiler=" + costoAlquiler + ", description=" + description ;

}

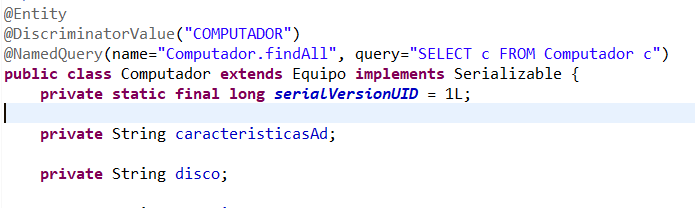
En la clase Computador

Luego de Entity indicamos el valor de la columna con los que se van a diferenciar los registros de la tabla Equipo

@DiscriminatorValue("COMPUTADOR")

Agregar el extends Equipo luego del nombre

Como clase extiende de Equipo, ya no va a necesitar el campo código



Agregar constructor parametrizado

public Computador(String codigo, String descripcion, String marca, float precio,String pantalla, String disco,

String memoria, String procesador,

String sistemaOperativo,String caracteristicasAd) {

super(codigo,descripcion,marca,precio);

this.caracteristicasAd = caracteristicasAd;

this.disco = disco;

this.memoria = memoria;

this.pantalla = pantalla;

this.procesador = procesador;

this.sistemaOperativo = sistemaOperativo;

}

Y el toString

@Override

public String toString() {

return "Computador" + super.toString() +", caracteristicasAd=" + caracteristicasAd + "]";

}

Clase Impresora

Primero crearemos un enum para el tipo de Impresion. En el paquete app.modelo crear :

public enum TipoImpresion {

LASER,TONER;

}

Así como en Computador hay que indicar el valor de la columna discriminante

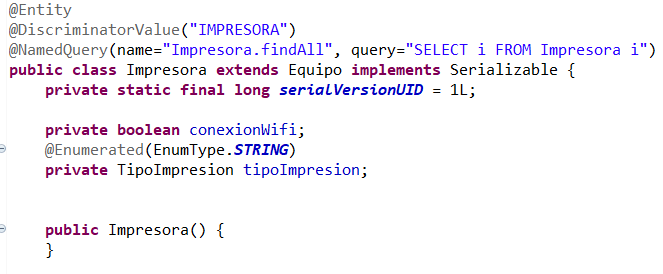
@DiscriminatorValue("IMPRESORA")

Dejaremos solo los atributos relacionados a la impresora de acuerdo al diagrama de clases

Cambiaremos el atributo tipoImpresion de String a TipoImpresion. Indicaremos que es un Enumerated:

@Enumerated(EnumType.STRING)

private TipoImpresion tipoImpresion;



Cambiaremos los setters & getters

Agregamos constructor parametrizado

public Impresora(String codigo, String descripcion, String marca, float precio, TipoImpresion tipoImpresion,boolean conexionWifi) {

super(codigo,descripcion,marca,precio);

this.conexionWifi = conexionWifi;

this.tipoImpresion = tipoImpresion;

}

Agregamos toString

@Override

public String toString() {

return "Impresora "+ super.toString()+ ",conexionWifi=" + conexionWifi + ", tipoImpresion=" + tipoImpresion + "]";

}

**PRUEBAS DEL MODELO IMPLEMENTADO**

Creamos paquete app.test

En el paquete creamos clase Test.

Crearemos método insertar para probar el ingreso de objetos a la base de datos

public static void insertarComputador() {

System.out.println("Insertar..");

EntityManagerFactory emf = Persistence.createEntityManagerFactory("JPATest2");

EntityManager em = emf.createEntityManager();

EntityTransaction et = em.getTransaction();

try {

//cambiar los datos para no tener duplicados

Computador com1 =new Computador("1001","Computador portatil 14''","LENOVO",70.0f,"14 \"","1TB","8GB","Intel i3","Windows 10","Camara web integrada, 3 puertos usb");

//Computador com1 =new Computador("1002","Computador portatil 14''","HP",60.0f,"14 \"","1TB","4GB","Intel i3","Windows 10","Camara web integrada, 3 puertos usb");

//Computador com1 =new Computador("1003","Computador de escritorio 15''","HP",70.0f,"14 \"","1TB","8GB","Intel i3","Windows 10","tarjeta grafica");

et.begin();

em.persist(com1);

et.commit();

} catch (Exception e) {

System.out.println("Error:" + e.getMessage());

} finally {

em.close();

}

}

public static void insertar() {

System.out.println("Insertar..");

EntityManagerFactory emf = Persistence.createEntityManagerFactory("JPATest2");

EntityManager em = emf.createEntityManager();

EntityTransaction et = em.getTransaction();

try {

//cambiar los datos para no tener duplicados

//impresoras

Impresora imp = new Impresora("1011","Impresora Laser multifuncion","HP",60.0f,TipoImpresion.LASER,true);

//Impresora imp = new Impresora("1012","Impresora multifuncion","EPSON",50.0f,TipoImpresion.TONER,false);

//Impresora imp = new Impresora("1013","Impresora Laser multifuncion","HP",55.0f,TipoImpresion.LASER,false);

et.begin();

em.persist(imp);

et.commit();

} catch (Exception e) {

System.out.println("Error:" + e.getMessage());

} finally {

em.close();

}

}

Consulta de datos

Podemos definir consultas a realizar sobre la tabla, agregándolas con un NamedQuery, por defecto se crea findAll

Utilizaremos este query para consultar los datos presentes en la tabla

public static void consultar() {

System.out.println("Consultar..");

EntityManagerFactory emf = Persistence.createEntityManagerFactory("JPATest2");

EntityManager em = emf.createEntityManager();

EntityTransaction et = em.getTransaction();

try {

et.begin();

TypedQuery<Computador> query = em.createNamedQuery("Computador.findAll",Computador.class);

List<Computador> lista = query.getResultList();

et.commit();

for(Computador c:lista) {

System.out.println(c);

}

} catch (Exception e) {

System.out.println("Error:" + e.getMessage());

} finally {

em.close();

}

}

**Agregando querys**

Agregaremos una nueva consulta para recuperar las computadoras que tengan deterninada marca. Los queries quedarian asi:

@NamedQueries({

@NamedQuery(name="Computador.findAll", query="SELECT c FROM Computador c"),

@NamedQuery(name="Computador.buscarMarca", query="SELECT c FROM Equipo e,Computador c WHERE e.codigo=c.codigo and e.marca=:marcaBuscar"),

})

En el metodo de busqueda hacer referencia a ese query y enviar el parametro

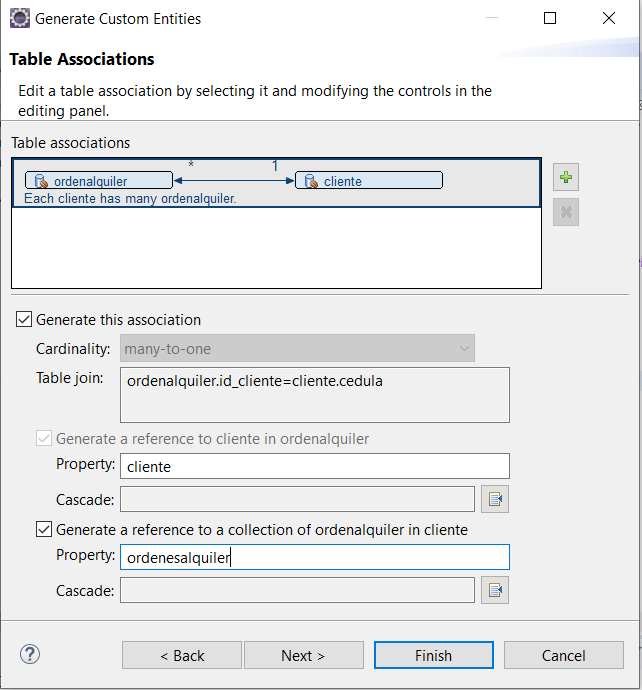
TypedQuery<Computador> query = em.createNamedQuery("Computador.buscarMarca",Computador.class);

query.setParameter("marcaBuscar", "HP");

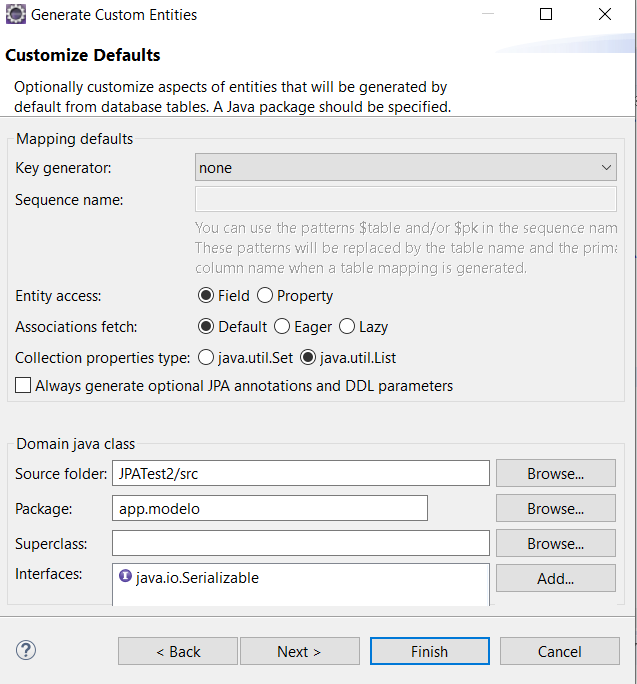
**RELACIÓN UNO A MUCHOS**

Sobre el paquete app.modelo damos click en New - JPA Entities from Tables. Seleccionamos las tablas cliente y ordenalquiler . Click en Next

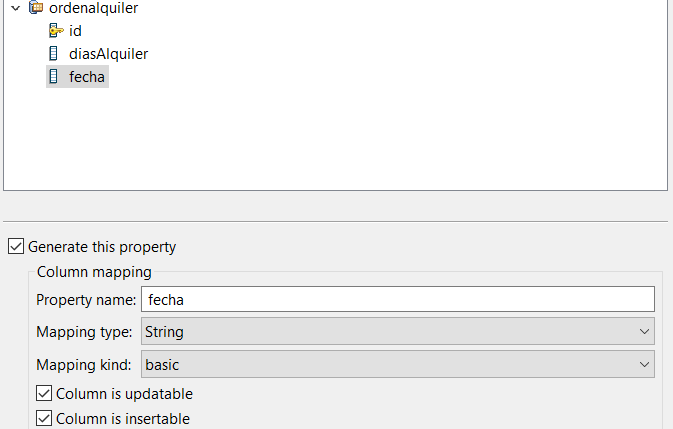
En la siguiente ventana seleccionamos la relación, aquí podemos cambiar los nombres de los atributros de las clases. Cambiaremos el nombre en el atributo para la clase Cliente : ordenesalquiler.



Luego click en Next



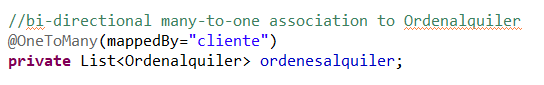
Click en Next . En la siguiente ventana vamos a customizar campo fecha para la tabla ordenalquiler



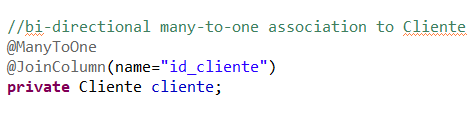
Click en Finish

Las nuevas clases se agregan

En la clase Cliente, vemos una lista de tipo Ordenalquiler



En la clase OrdenAlquiler vemos el atributo de la clase Cliente



En la base de datos la orden tiene un id autoincrement. En la clase agregar la notacion @GeneratedValue al id

@Id @GeneratedValue(strategy=GenerationType.IDENTITY)

private int id;

Agregamos un constructor parametrizado a la clase Ordenalquiler

public Ordenalquiler(String fecha, short diasAlquiler, Cliente cliente) {

this.diasAlquiler = diasAlquiler;

this.fecha = fecha;

this.cliente = cliente;

}

Agregamos también el método toString

En la clase Test.java, creamos un método para insertar una orden

En este método se va a buscar un cliente que ya existe para poder tener el objeto Cliente.

La fecha se va a transformar en un String usando SimpleDateFormat

public static void insertarOrden() {

System.out.println("Insertar orden..");

EntityManagerFactory emf = Persistence.createEntityManagerFactory("JPATest2");

EntityManager em = emf.createEntityManager();

EntityTransaction et = em.getTransaction();

try {

//buscar clliente

Cliente cliente = em.find(Cliente.class, "4444444");

SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

String fecha =sdf.format(new Date());

short dias=4;

Ordenalquiler orden = new Ordenalquiler(fecha,dias,cliente);

et.begin();

em.persist(orden);

et.commit();

} catch (Exception e) {

System.out.println("Error:" + e.getMessage());

} finally {

em.close();

}

}

Luego de llamar al método en nuevo registro en la base de datos es insertado.

También podemos consultar las ordenes que tiene un determinado cliente

public static void consultarOrdenesCliente() {

System.out.println("consultar orden de cliente..");

EntityManagerFactory emf = Persistence.createEntityManagerFactory("JPATest2");

EntityManager em = emf.createEntityManager();

try {

//buscar clliente

Cliente cliente = em.find(Cliente.class, "4444444");

//recuperar sus ordenes y mostrar

for (Ordenalquiler orden:cliente.getOrdenesalquiler()) {

System.out.println(orden);

}

} catch (Exception e) {

System.out.println("Error:" + e.getMessage());

} finally {

em.close();

}

}

Queries personalizados se pueden crear también fuera de la entidad. Para esto hay que usar el método createQuery.

El siguiente método muestra las computadoras con precio menor a 80

public static void consultarPorPrecio() {

System.out.println("Consultar por precio..");

EntityManagerFactory emf = Persistence.createEntityManagerFactory("JPATest3");

EntityManager em = emf.createEntityManager();

EntityTransaction et = em.getTransaction();

try {

String sQuery ="SELECT c FROM Computador c, Equipo e WHERE e.codigo = c.codigo and e.costoAlquiler<80 ";

et.begin();

List<Computador> lista = em.createQuery(sQuery).getResultList();

et.commit();

for(Computador c:lista) {

System.out.println(c);

}

} catch (Exception e) {

System.out.println("Error:" + e.getMessage());

} finally {

em.close();

}

}

Links relacionados

<https://thoughts-on-java.org/many-relationships-additional-properties/>

<https://tomee.apache.org/examples-trunk/jpa-enumerated/>

<https://thoughts-on-java.org/jpa-generate-primary-keys/>