Ministerio de Producción
Presidencia de la Nación

Ministerio de Educación y Deportes

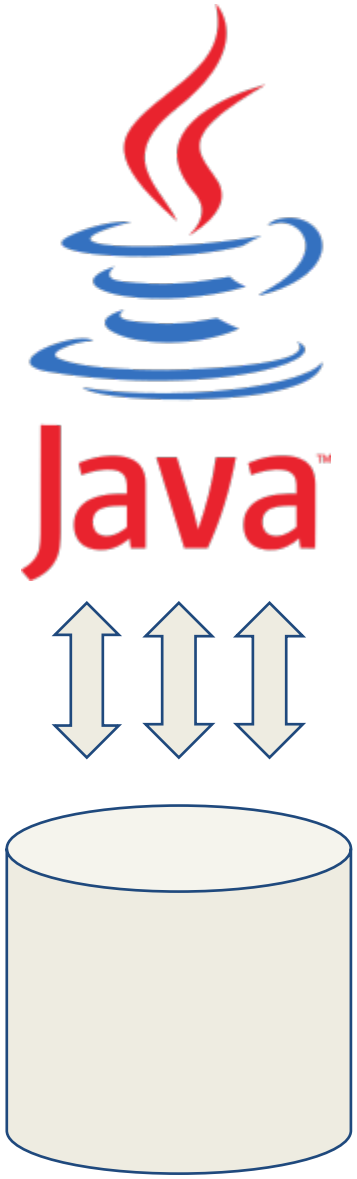
Subsecretaría de Servicios Tecnológicos y Productivos

Programa 111 mil
VOS PODÉS SER UNO.

HIBERNATE

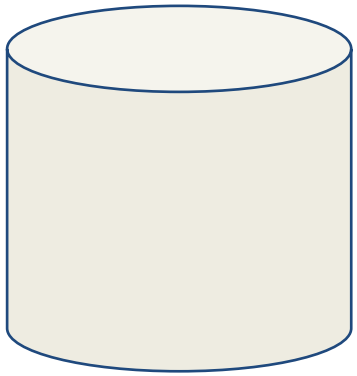
Agenda

- Java y bases de datos
- JDBC
- Mapeo Objeto-Relacional
- Hibernate
- Configurar Hibernate
- Mapear clases a tablas
- Consultas



Problema

- ¿Cómo almacenamos los datos que usa nuestra aplicación Java?
- ¿Qué hacemos con los datos almacenados en una base de datos?



JDBC

- JDBC es una API (Application Programming Interface) que permite acceder a diversas bases de datos de una forma estándar.
- Cada proveedor de bases de datos debe escribir su propio Driver JDBC.
- El acceso a la base de datos se realiza mayoritariamente utilizando consultas SQL.
- Requiere mucho código y el resultado final queda ligado a las sentencias SQL de motor original.

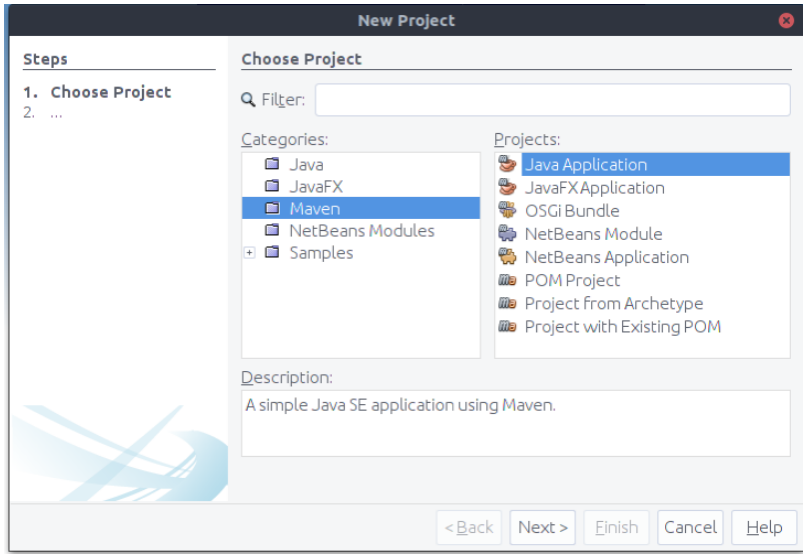
Ejemplo JDBC

Consideremos el siguiente script de creación de base de datos MySql.

```
create database db111mil;
create user 'user111mil'@'localhost' identified by '111mil';
grant all on db111mil.* to 'user111mil'@'localhost';
CREATE TABLE `E01_CLIENTE` (
     `nro_cliente` int(11) NOT NULL,
     `apellido` varchar(45) NOT NULL,
     `nombre` varchar(45) NOT NULL,
     `direccion` varchar(45) NOT NULL,
     `activo` bit(1) NOT NULL,
     PRIMARY KEY (`nro_cliente`)
);
```

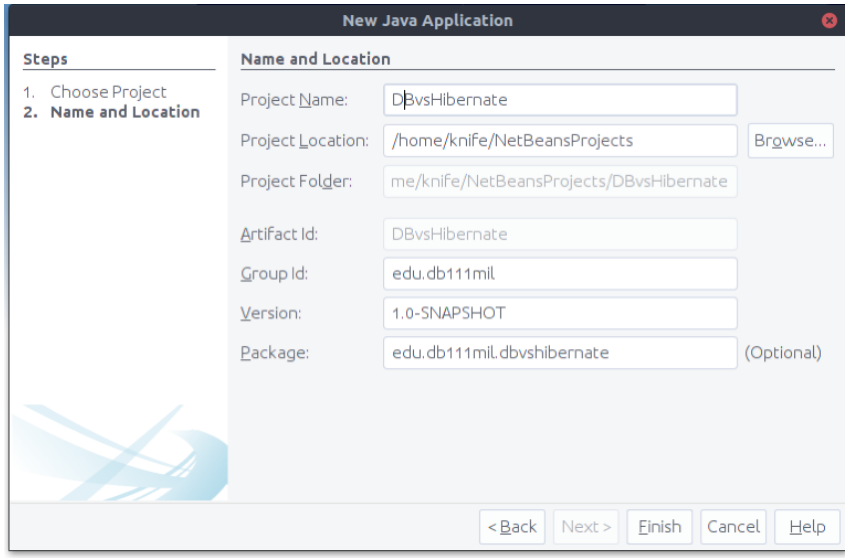
Ejemplo JDBC

- Creamos un nuevo proyecto en NetBeans.
- En vez de seleccionar Java en “Categories”, seleccionamos Maven y en “Project” seleccionamos “Java Application”
- Maven es programa que ayuda en el proceso de compilación. Además permite agregar dependencias (en este caso el driver JDBC para MySql) muy fácilmente.



Ejemplo JDBC

- Project Name: Nombre del proyecto.
- Project Location: Dónde se almacena el proyecto.
- GroupId: Nombre de paquete que utiliza nuestra organización.
- Version: Versión actual del producto.
- Package: Paquete de nuestro proyecto. Si se deja el por defecto es GroupId.“Project Name” todo en minúscula.



Ejemplo JDBC

- En el proyecto recién creado aparece una carpeta llamada “Project Files”.
- Dentro de esta carpeta hay un archivo llamado “pom.xml”
- Este archivo es el archivo que utiliza Maven para saber como construir nuestro proyecto.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>edu.db111mil</groupId>
    <artifactId>DBvsHibernate</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
</project>
```

Ejemplo JDBC

Tenemos que agregar la dependencia del driver.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>edu.db111mil</groupId>
    <artifactId>DBvsHibernate</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>6.0.5</version>
        </dependency>
    </dependencies>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
</project>
```

Ejemplo JDBC

Para el ejemplo vamos a desarrollar una main que muestra a todos los clientes presentes en la base de datos.

```
try {
     String url = "jdbc:mysql://localhost:3306/db111mil";
     Connection conn = DriverManager.getConnection(url,
               "user111mil",
               "111mil");
     PreparedStatement st = conn.prepareStatement("SELECT * FROM E01_CLIENTE");
     ResultSet resultSet = st.executeQuery();
     while (resultSet.next()) {
          int id = resultSet.getInt("nro_cliente");
          String nombre = resultSet.getString("nombre");
          String apellido = resultSet.getString("apellido");
          String direccion = resultSet.getString("direccion");
          boolean activo = resultSet.getBoolean("activo");
          System.out.println("" + id + ", " + nombre + ", " +
                    apellido + ", " + direccion + ", " + activo);
     }
} catch (SQLException ex) {
          ex.printStackTrace();
}
```

Mapeo Objeto Relacional (ORM)

- Los ORM tiene como objetivo abstraer al desarrollado de las cuestiones particulares de las bases de datos.
- Transforman de forma automática los objetos a tuplas que se persisten en la base de datos cuando llega el momento de almacenarlos.
- Transforman de forma automática las tuplas persistidas a objetos. De esta manera se puede operar con los datos.
- Evitan el uso de SQL directo. Esto es una ventaja porque las diferentes bases de datos tiene diferencias en cómo está implementado SQL.

Hibernate

- Es un ORM para Java.
- Para configurar Hibernate se utilizan archivos XML.
- Las tablas son mapeadas a clases POJO (Plain Old Java Objects), es decir, clases con métodos getters y setters.
- El mapeo tabla-objetos se define a través de archivos XML.
- Posee su propio lenguaje de consultas.

Hibernate: Ejemplo

- Para el ejemplo se escribirá un programa que permita hacer la misma consulta a la base de datos.
- Para realizar un proyecto Hibernate es recomendable utilizar Maven.
- Algo importante a tener en cuenta es que Hibernate utiliza JDBC para conectarse a la base de datos. Por este motivo las dependencias de JDBC que utilizamos en el proyecto anterior van a estar presentes.
- También hay que agregar la dependencia de Hibernate. Hibernate es una librería con muchos componentes, para este curso solo trabajaremos con el componente Core (núcleo o principal).

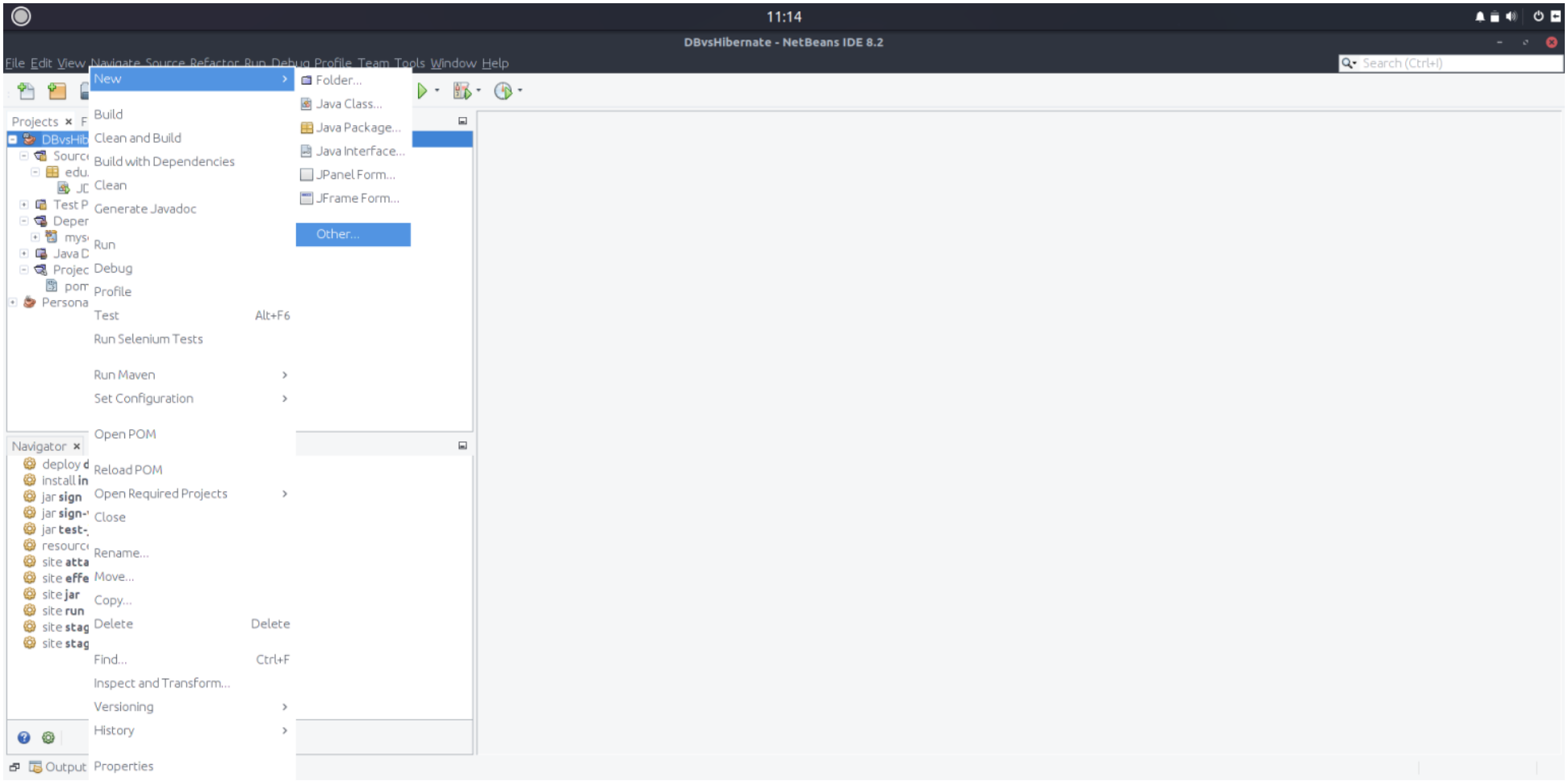
Hibernate: pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>edu.db111mil</groupId>
    <artifactId>DBvsHibernate</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>6.0.5</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.2.8.Final</version>
        </dependency>
    </dependencies>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
</project>
```

Hibernate: Configurar

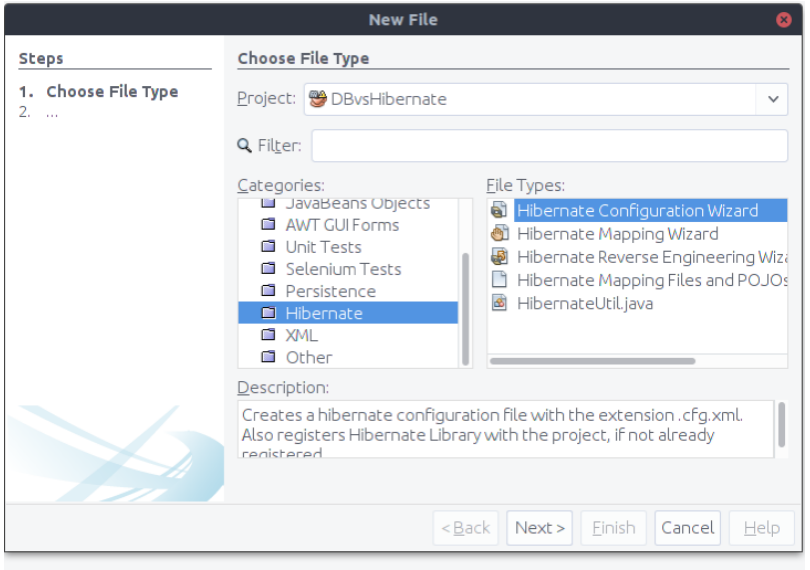
- A diferencia de JDBC, la configuración de la conección a la base de datos se realiza en otro archivo llamado hibernate.conf.xml.
- NetBeans tiene soporte para generar este archivo de forma sencilla.
- El primer paso es hacer clic secundario sobre el paquete y seleccionar New > Other.

Hibernate: Configurar



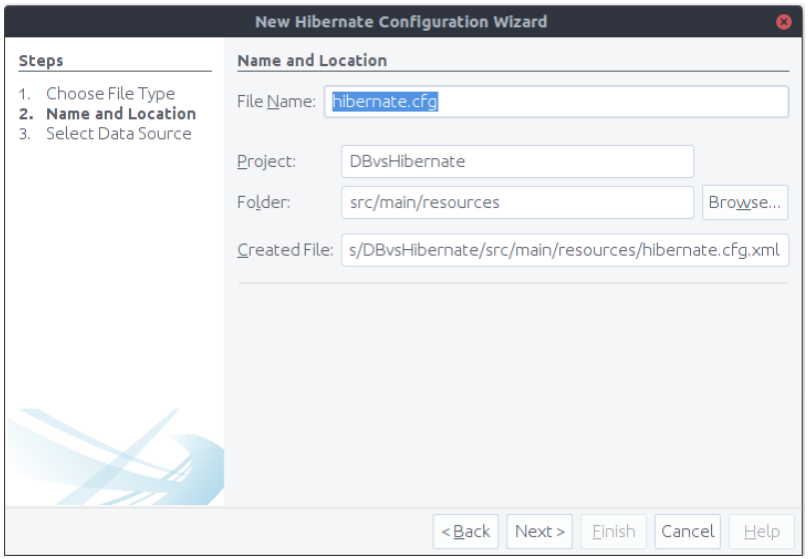
Hibernate: Configurar

- En “Category” seleccionar “Hibernate”.
- En “File type” seleccionar “Hibernate Configuration Wizard”(Asistente para la configuración de Hibernate).
- Hacer clic en “Next >”



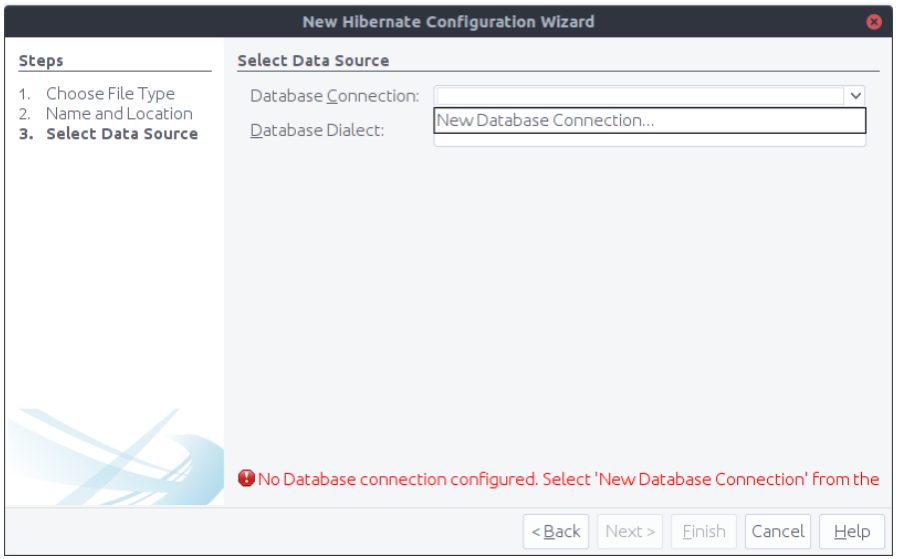
Hibernate: Configurar

- En el segundo paso del asistente, los valores por defecto son lo que utilizaremos.
- Se puede ver el nombre del archivo, a qué proyecto está asociado y dónde se creará.
- Hacer clic en “Next >”



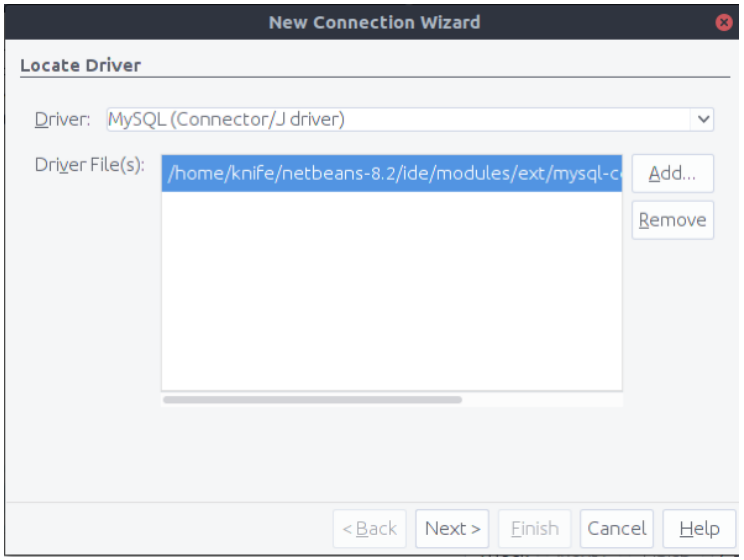
Hibernate: Configurar

- En esta ventana hay que seleccionar la conexión a base de datos usaremos.
- Como no hay conexiones existentes debemos seleccionar “New Database Connection…” (nueva conexión a una base de datos)



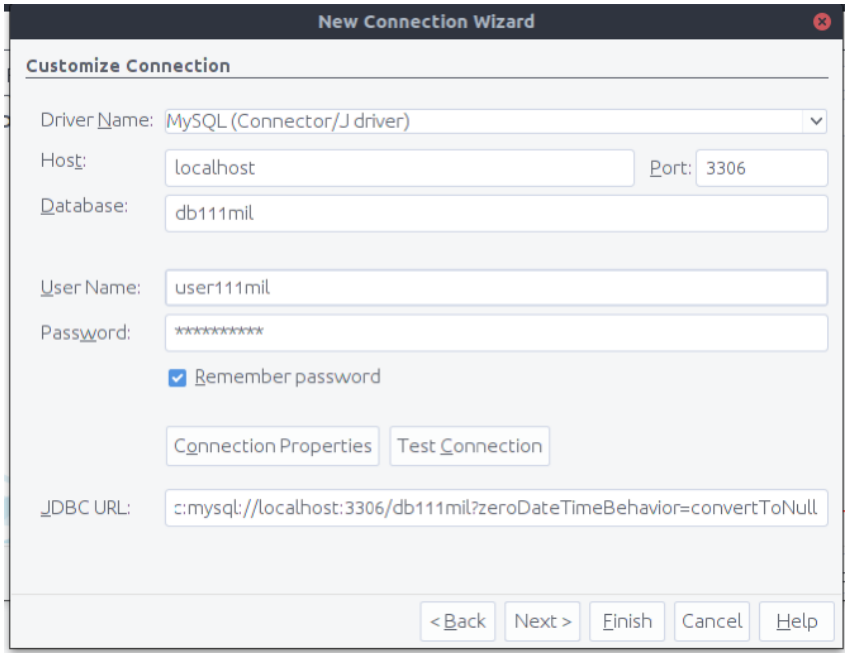
Hibernate: Configurar

- El primer paso consiste en seleccionar el Driver JDBC.
- Por defecto, debería estar seleccionado el driver MySql ya que es el único driver que Maven bajo.
- En caso de no estar seleccionado, seleccione el Driver “MySql (Connector/JDriver)”



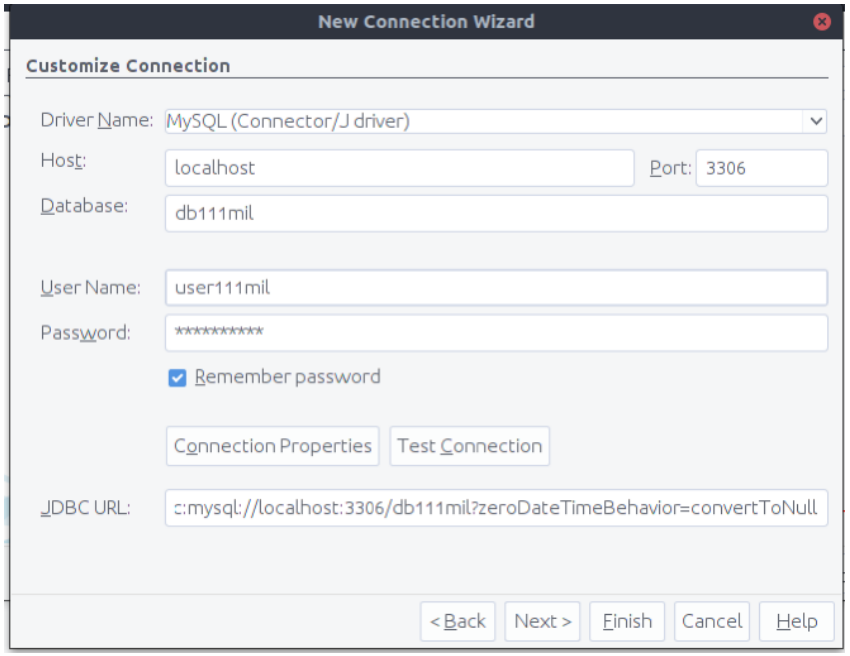
Hibernate: Configurar

- En el siguiente paso debemos configurar la conexión a la base de datos.
- Host: localhost (indicamos que la base de datos corre en nuestra PC).
- Port: 3306 (es el por defecto de MySql.



Hibernate: Configurar

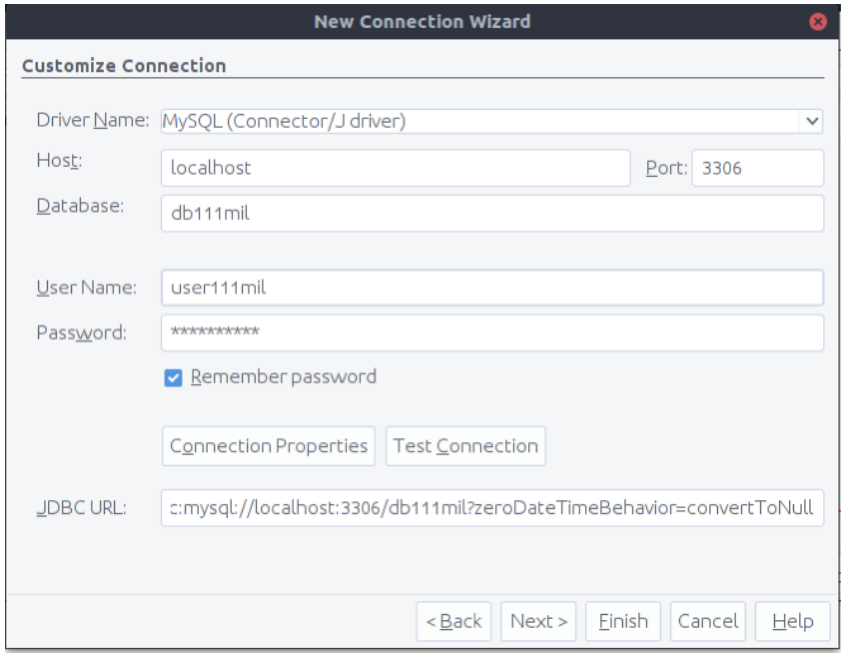
- Database: el nombre de la base de datos. En este caso db111mil.
- User Name: nombre de usuario. En este caso user111mil.
- Password: en este caso 111mil.
- JDBC URL: se genera automaticamente. Hasta el signo de pregunta es la misma URL del ejemplo anterior.



Hibernate: Configurar

Antes de continuar se puede verificar que la conexión a la base de datos funcione.
Para esto, se debe hacer clic en “Test Connection”.

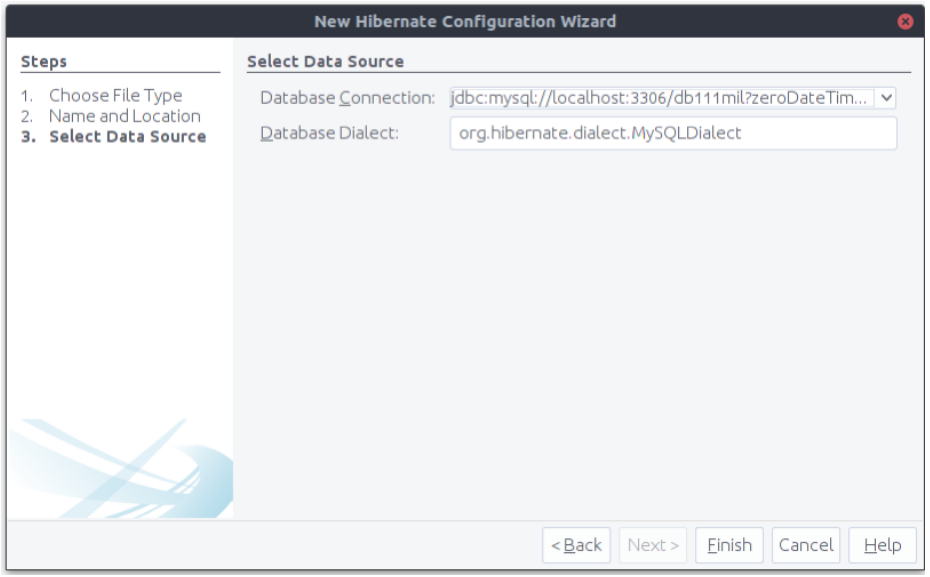
En caso de existir problemas, debemos verificar la configuración.



Hibernate: Configurar

Finalmente, hacer clic en “Finish >” para generar el archivo hibernate.conf.xml.

El archivo generado aparecerá en el proyecto dentro de “Other Sources” > “scr/main/resources” > “default package”



Hibernate: hibernate.conf.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<!--
...
-->
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">
        jdbc:mysql://localhost:3306/db111mil?zeroDateTimeBehavior=convertToNull
    </property>
    <property name="hibernate.connection.username">user111mil</property>
    <property name="hibernate.connection.password">111mil</property>
  </session-factory>
</hibernate-configuration>
```

Hibernate: Mapear una tabla

- Definir una clase para mapear la entidad. Definimos la clase edu.db111mil.hibernate.dbvshibernate.hibernate.Cliente
- Para cada atributo de la entidad definir un atributo de clase. Por simplicidad, le asignamos el mismo nombre.
- Para cada tipo de de dato en la base de datos, analizar con qué tipo de dato Java se corresponde.
- Definir getters/setters y demás métodos de la clase.
- Definir un constructor sin parámetros (constructor por defecto).
- Definir el archivo xml de mapeo Entidad-Clase.
- Agregar el mapeo el archivo de mapeo a hibernate.conf.xml (si el archivo de mapeo se construye usando NetBeans este paso es automático).

Hibernate: Mapear una table

Mapping type	Java type	ANSI SQL Type
integer	int or java.lang.Integer	INTEGER
long	long or java.lang.Long	BIGINT
short	short or java.lang.Short	SMALLINT
float	float or java.lang.Float	FLOAT
double	double or java.lang.Double	DOUBLE
big_decimal	java.math.BigDecimal	NUMERIC
character	java.lang.String	CHAR (1)
string	java.lang.String	VARCHAR
byte	byte or java.lang.Byte	TINYINT
boolean	boolean or java.lang.Boolean	BIT
yes/no	boolean or java.lang.Boolean	CHAR(1) (‘Y’ or ‘N’)
true/false	boolean or java.lang.Boolean	CHAR(1) (‘T’ or ‘F’)

Hibernate: Mapear una table

Mapping type	Java type	ANSI SQL Type
date	java.util.Date or java.sql.Date	DATE
time	java.util.Date or java.sql.Time	TIME
timestamp	java.util.Date or java.sql.Time	TIMESTAMP
calendar	java.util.Calendar	TIMESTAMP
calendar_date	java.util.Calendar	DATE

Hibernate: Mapear una tabla

Mapping type	Java type	ANSI SQL Type
binary	byte[]	VARBINARY (or BLOB)
text	java.lang.String	CLOB
serializable	Clases que implementen java.io.Serializable	VARBINARY (or BLOB)
blob	java.sql.Clob	BLOB
clob	java.sql.Blob	CLOB

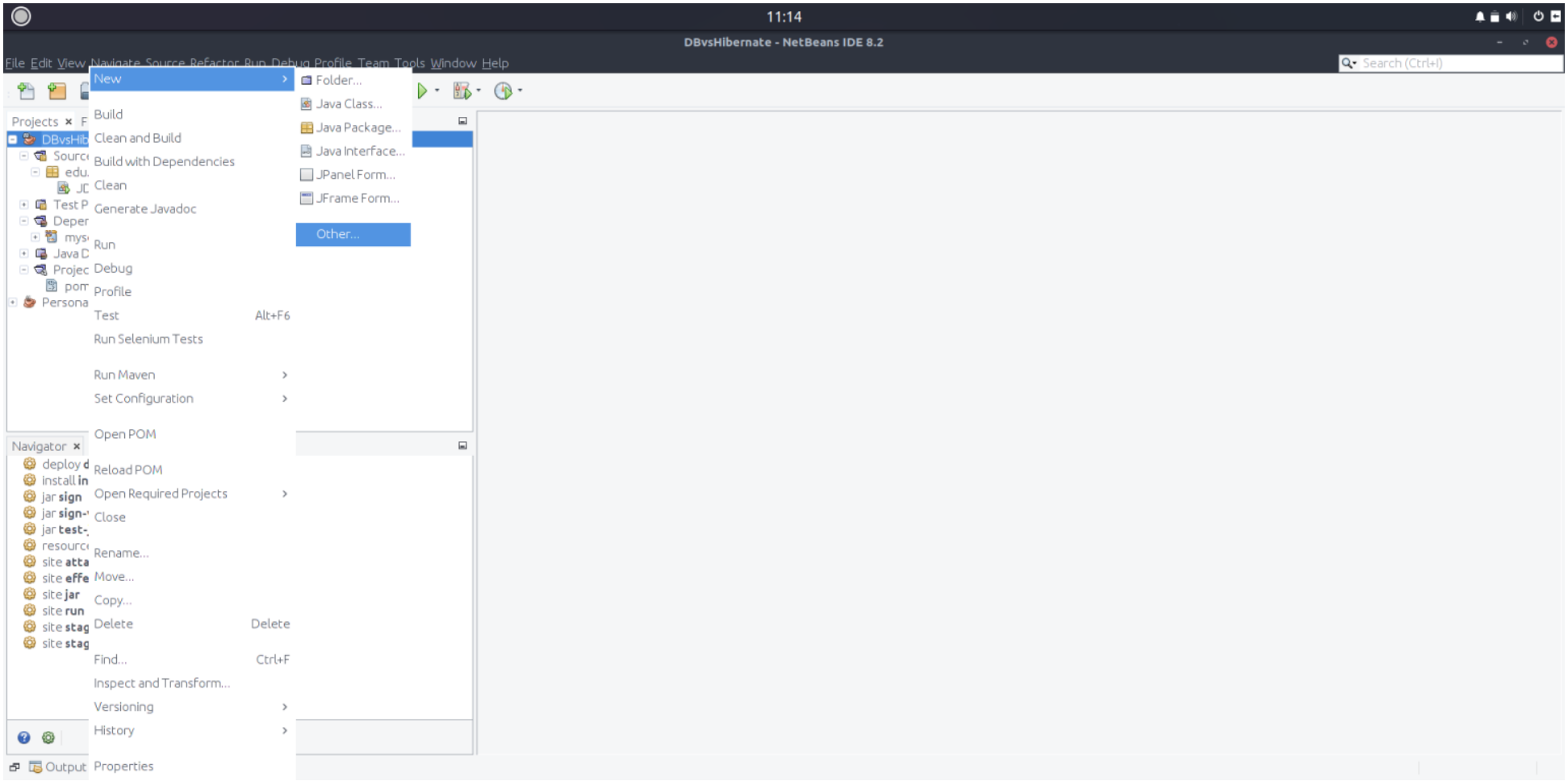
Hibernate: Mapear una tabla- Clase

```
package edu.db111mil.dbvshibernate.hibernate;

public class Cliente {

    private int nroCliente;
    private String nombre;
    private String apellido;
    private String direccion;
    private boolean activo;

    public Cliente(){
        super();
    }

    public int getNroCliente() {
        return nroCliente;
    }

    public void setNroCliente(int nroCliente) {
        this.nroCliente = nroCliente;
    }
    ...// Getters y setters
}
```

Hibernate: XML Mapeo

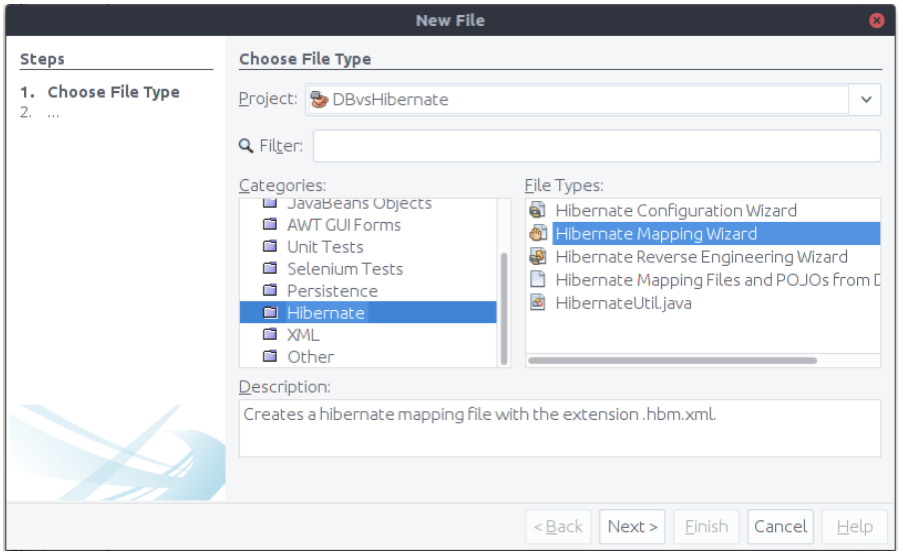
- Crear el archivo Cliente.hbm.xml en el mismo directorio que hibernate.conf.xml.
- Definir qué clase va a mapear que tabla.
- Definir cuál es el id (clave primaria).
- Definir cuales son las propierty (atributos - columna).
- No presente en este ejemplo:
  - Generador de Id.
  - Relaciones:
    - One-to-One (uno a uno).
    - Many-to-One (muchos a uno).
    - One-to-Many (uno a mucho).
    - Many-to-Many (muchos a muchos).

Hibernate: XML Mapeo



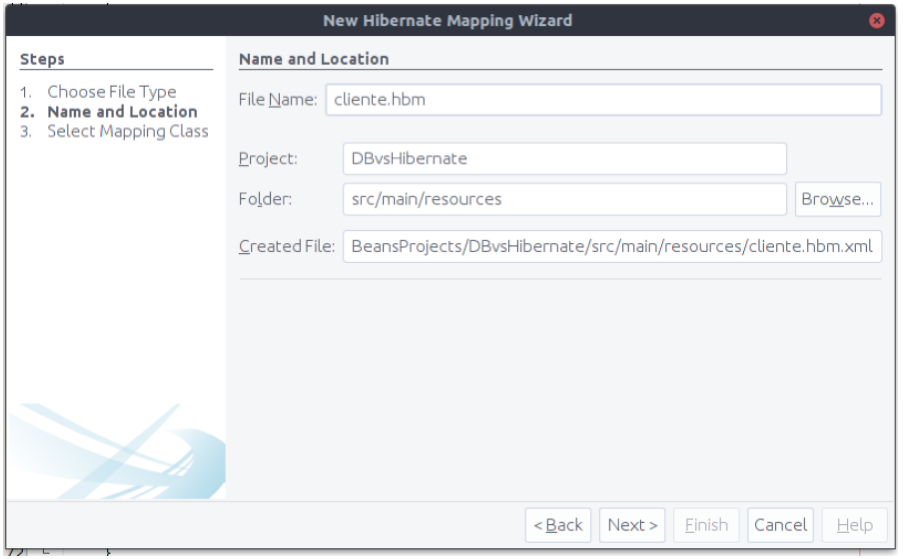
Hibernate: XML Mapeo

- En “Category” seleccionar “Hibernate”.
- En “File type” seleccionar “Hibernate Mapping Wizard”(Asistente para el mapeo de Hibernate).
- Hacer clic en “Next >”



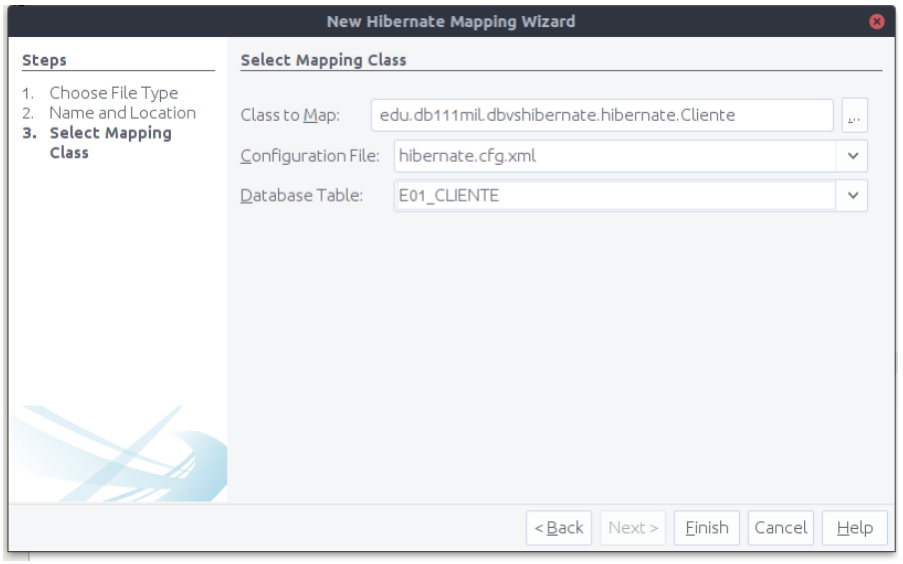
Hibernate: XML Mapeo

- En el campo “File Name” escribir el nombre del archivo a generar.
- Se recomienda: nombre_de_clase.hbm
- Los otros campos no se deben cambiar.
- Hacer clic en “Next >”



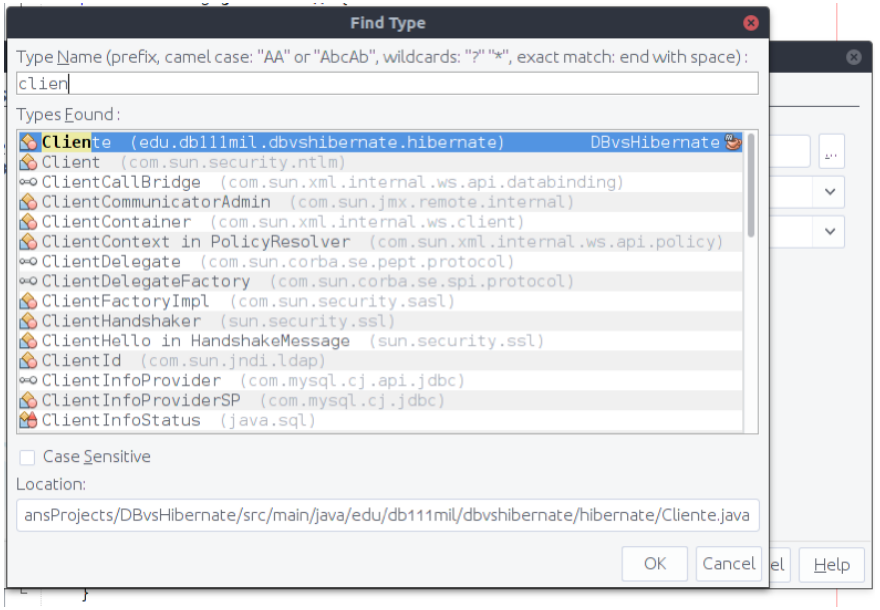
Hibernate: XML Mapeo

- Class to Map: Seleccionar la clase a mapear. En este caso Cliente.
- Configuration File: el nombre del archivo de configuración de hibernate. Definido previamente.
- Database Table: Seleccionar la tabla a mapear. En este caso, E01-CLIENTE.



Hibernate: XML Mapeo

- En la ventana anterior, presionando sobre "..." a la derecha del campo "Class to Map" aparece un asistente para seleccionar la clase.
- Con solo escribir parte del nombre de la clase, el sistema filtra las clases posibles.



Hibernate: XML Mapeo

El archivo generado se llama cliente.hbm.xml, de momento solo define que clase mapea que tabla.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- ...-->
<hibernate-mapping>
    <class name="edu.db111mil.dbvshibernate.hibernate.Cliente"  table="E01_CLIENTE"></class>
</hibernate-mapping>
```

Hibernate: XML Mapeo

Para mapear los atributos hay que agregar los siguiente.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- ...-->
<hibernate-mapping>
    <class name="edu.db111mil.dbvshibernate.hibernate.Cliente" table="E01_CLIENTE">
        <id name="nroCliente" column="nro_cliente"/>
        <property name="nombre"/>
        <property name="apellido"/>
        <property name="telefono"/>
        <property name="direccion"/>
        <property name="activo"/>
    </class>
</hibernate-mapping>
```

Hibernate: XML Mapeo

El asistente automáticamente agrega el tag mapping al hibernate.conf.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<!--... -->
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/db111mil?zeroDateTimeBehavior=convertToNull
</property>
    <property name="hibernate.connection.username">user111mil</property>
    <property name="hibernate.connection.password">111mil</property>
    <mapping resource="cliente.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Hibernate: XML Mapeo

Buscando a nuestro cliente en la BD:

```
public static void main(String[] args){
        StandardServiceRegistry registry = new StandardServiceRegistryBuilder()
            .configure() // obtiene los valores de hibernate.cfg.xml
            .build();
        try {
            SessionFactory sessionFactory = new MetadataSources(registry)
                    .buildMetadata().buildSessionFactory();
            Session session = sessionFactory.openSession();

            session.beginTransaction();
            Cliente c = session.load(Cliente.class, 1);
            System.out.println(c);


            session.getTransaction().commit();
            session.close();
            sessionFactory.close();
        }
        catch (Exception e) {
            ...
        }
    }
```

Hibernate: Cargando la configuración

El siguiente código inicializa Hibernate y debe ser ejecutado solo una vez, es decir, solo se debe ejecutar la primera vez que se utiliza Hibernate en el programa.

```
StandardServiceRegistry registry = new StandardServiceRegistryBuilder()
    .configure() // obtiene los valores de hibernate.cfg.xml
    .build();
SessionFactory sessionFactory = new MetadataSources(registry)
    .buildMetadata().buildSessionFactory();
```

Hibernate: Session

La sesión es la forma de acceder a la base de datos. Se pueden crear todas las que se necesiten durante el curso del programa, usando el “SessionFactory” creado con el código de inicialización. Dentro de la sesión hay transacciones.

```
Session session = sessionFactory.openSession();
session.beginTransaction();
....
session.getTransaction().commit();
session.beginTransaction();
....
session.getTransaction().commit();
session.close();
```

Hibernate: Finalizar SessionFactory

Finalmente, al finalizar el programa es importante cerrar el SessionFactory. En otro caso, el programa no finalizará correctamente.

```
sessionFactory.close();
```

Hibernate: XML Mapeo

Accediendo al cliente con clave 1:

```
Cliente c = session.load(Cliente.class, 1);
System.out.println(c);
```

Hibernate: XML Mapeo

Accediendo al cliente con clave 1:

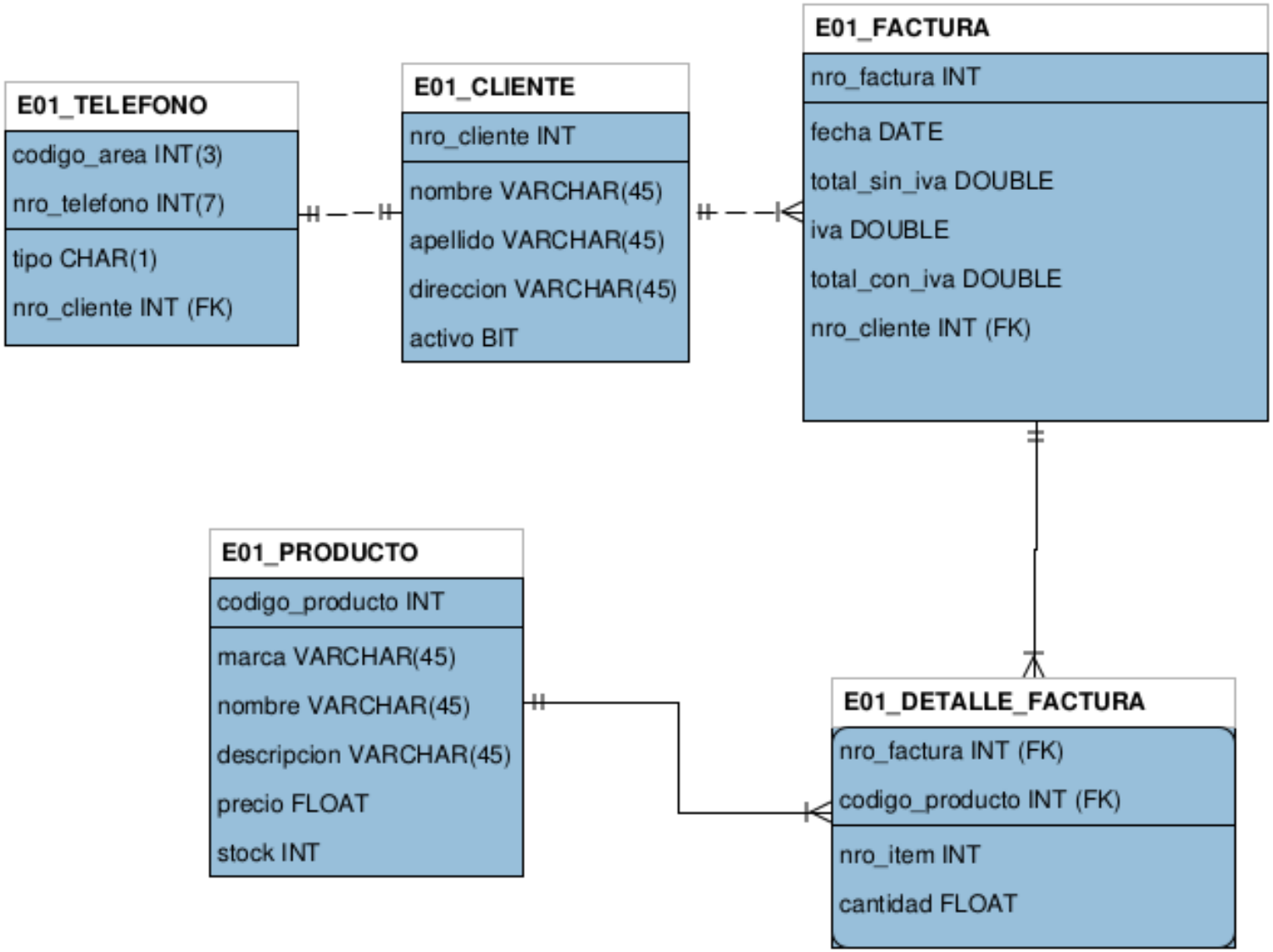
```
Cliente c = session.load(Cliente.class, 1);
System.out.println(c);
```

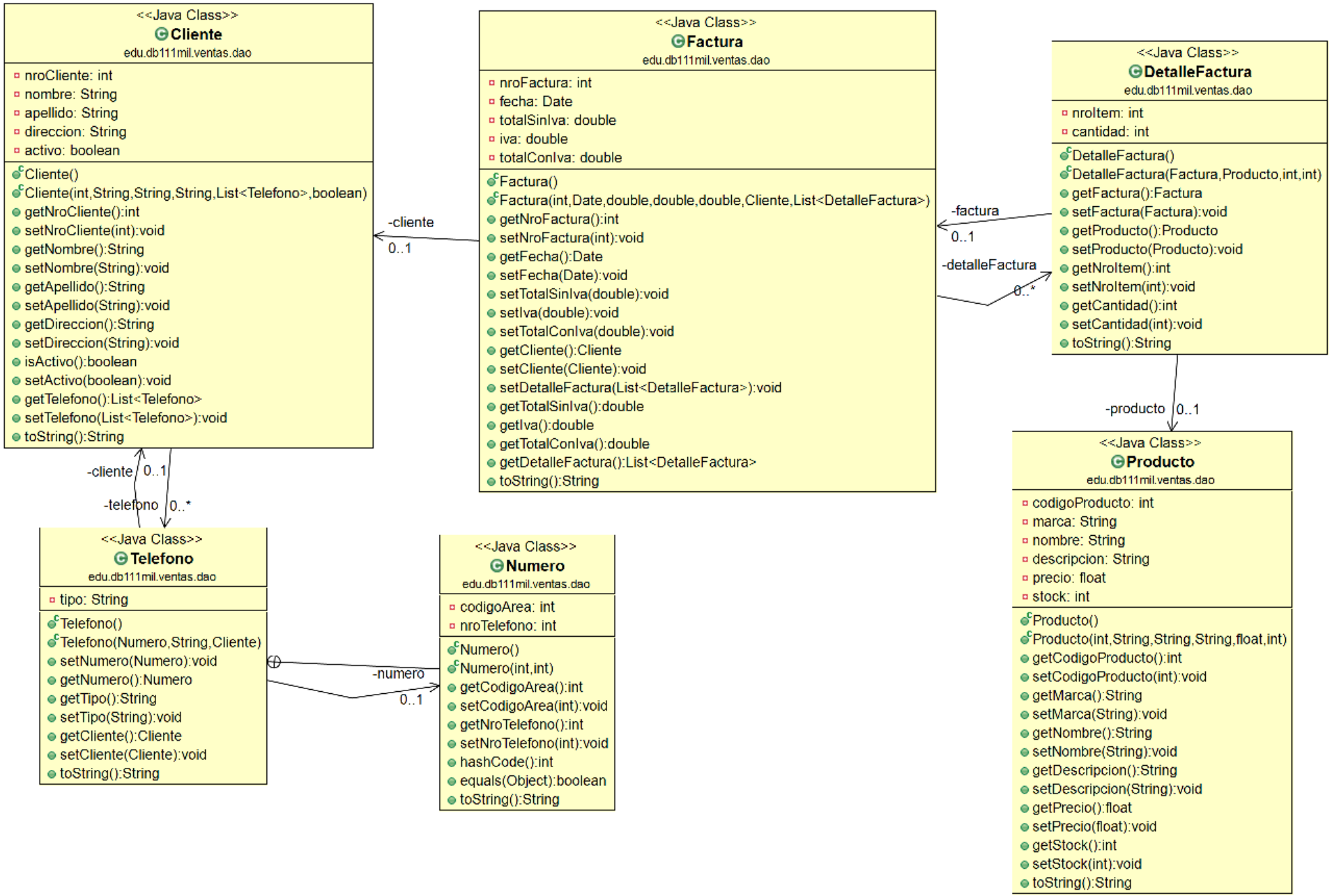
Pero… ¿Esto es suficiente para hacer un programa?
No, en las siguientes slides veremos cómo mapear relaciones, realizar queries y actualizar objetos.

Hibernate: Ejemplo Completo

En las siguientes slides vamos a realizar un ejemplo completo para mapear correctamente la base de datos que hemos utilizado hasta el momento a un modelo de objetos compatible con hibernate.
Puntos importantes:

- Se tendrán que hacer concesiones del diseño POO para mapear perfectamente la base de datos.
- En muchos caso, se verá doble navegación (Clase A -> Clase B y vise-versa). Esto puede no tener sentido, pero es para ilustrar diferentes tipos de relaciones.
- Se verán formas distintas de hacer lo mismo. En un proyecto real, se recomienda ser lo más consistente posible.


E01_TELEFONO
codigo_area INT(3)
nro_telefono INT(7)
tipo CHAR(1)
nro_cliente INT (FK)
E01_CLIENTE
nro_cliente INT
nombre VARCHAR(45)
apellido VARCHAR(45)
direccion VARCHAR(45)
activo BIT
E01_FACTURA
nro_factura INT
fecha DATE
total_sin_iva DOUBLE
iva DOUBLE
total_con_iva DOUBLE
nro_cliente INT (FK)
E01_PRODUCTO
codigo_producto INT
marca VARCHAR(45)
nombre VARCHAR(45)
descripcion VARCHAR(45)
precio FLOAT
stock INT
E01_DETALLE_FACTURA
nro_factura INT (FK)
codigo_producto INT (FK)
nro_item INT
cantidad FLOAT


<<Java Class>>
Cliente
edu.db111mil.ventas.dao
nroCliente: int
nombre: String
apellido: String
direccion: String
activo: boolean
Cliente()
Cliente(int,String,String,String,List<Telefono>,boolean)
getNroCliente():int
setNroCliente(int):void
getNombre():String
setNombre(String):void
getApellido():String
setApellido(String):void
getDireccion():String
setDireccion(String):void
isActivo():boolean
setActivo(boolean):void
getTelefono():List<Telefono>
setTelefono(List<Telefono>):void
toString():String
<<Java Class>>
Factura
edu.db111mil.ventas.dao
nroFactura: int
fecha: Date
totalSinIva: double
iva: double
totalConIva: double
Factura()
Factura(int,Date,double,double,double,Cliente,List<DetalleFactura>)
getNroFactura():int
setNroFactura(int):void
getFecha():Date
setFecha(Date):void
setTotalSinIva(double):void
setIva(double):void
setTotalConIva(double):void
getCliente():Cliente
setCliente(Cliente):void
setDetalleFactura(List<DetalleFactura>):void
getTotalSinIva():double
getIva():double
getTotalConIva():double
getDetalleFactura():List<DetalleFactura>
toString():String
<<Java Class>>
DetalleFactura
edu.db111mil.ventas.dao
nroItem: int
cantidad: int
DetalleFactura()
DetalleFactura(Factura,Producto,int,int)
getFactura():Factura
setFactura(Factura):void
getProducto():Producto
setProducto(Producto):void
getNroItem():int
setNroItem(int):void
getCantidad():int
setCantidad(int):void
toString():String
<<Java Class>>
Producto
edu.db111mil.ventas.dao
codigoProducto: int
marca: String
nombre: String
descripcion: String
precio: float
stock: int
Producto()
Producto(int,String,String,String,float,int)
getCodigoProducto():int
setCodigoProducto(int):void
getMarca():String
setMarca(String):void
getNombre():String
setNombre(String):void
getDescripcion():String
setDescripcion(String):void
getPrecio():float
setPrecio(float):void
getStock():int
setStock(int):void
toString():String
<<Java Class>>
Telefono
edu.db111mil.ventas.dao
tipo: String
Telefono()
Telefono(Numero,String,Cliente)
setNumero(Numero):void
getNumero():Numero
getTipo():String
setTipo(String):void
getCliente():Cliente
setCliente(Cliente):void
toString():String
<<Java Class>>
Numero
edu.db111mil.ventas.dao
codigoArea: int
nroTelefono: int
Numero()
Numero(int,int)
getCodigoArea():int
setCodigoArea(int):void
getNroTelefono():int
setNroTelefono(int):void
hashCode():int
equals(Object):boolean
toString():String
-cliente
0..1
-factura
0..1
-detalleFactura
0..*
-producto
0..1
-cliente
0..1
-telefono
0..*
-numero
0..1

Hibernate: Clase Telefono

```
public class Telefono {

    static class Numero implements Serializable {
        private int codigoArea;
        private int nroTelefono;

        public Numero() {
        }

        public Numero(int codigoArea, int nroTelefono) {
            this.codigoArea = codigoArea;
            this.nroTelefono = nroTelefono;
        }

        public int getCodigoArea() {
            return codigoArea;
        }

        public void setCodigoArea(int codigoArea) {
            this.codigoArea = codigoArea;
        }
        ...
```

Hibernate: Clase Telefono.Numero

```java
@Override
public boolean equals(Object obj) {
     if (this == obj) {
         return true;
     }
     if (obj == null) {
         return false;
    }
    if (getClass() != obj.getClass()) {
         return false;
    }
    final Numero other = (Numero) obj;
    if (this.codigoArea != other.codigoArea) {
         return false;
    }
    if (this.nroTelefono != other.nroTelefono) {
         return false;
    }
    return true;
}
```

```java
@Override
public int hashCode() {
    int hash = 5;
    hash = 29 * hash + this.codigoArea;
    hash = 29 * hash + this.nroTelefono;
    return hash;
}
```

Hibernate: Clase Telefono.Numero

```
    @Override
    public String toString() {
        return "Numero{" + "codigoArea=" + codigoArea + ", nroTelefono=" + nroTelefono + '}';
    }

}

private Numero numero;
private String tipo;
private Cliente cliente;
… //Resto de la definición de la Clase Telefono
```

Hibernate: Clase Telefono

```
public class Telefono {

    static class Numero implements Serializable {
       ....
    }

    private Numero numero;
    private String tipo;
    private Cliente cliente;

    public Telefono() {
    }

    public Telefono(Numero numero, String tipo, Cliente cliente) {
        this.numero = numero;
        ...
    }

    public void setNumero(Numero numero) {
        this.numero = numero;
    }

    public Numero getNumero() {
        return numero;
    }
    //Getters, Setters y toString (Sacar el cliente del toString para evitar impresión circular
}
```

Hibernate: Mapeo telefono.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping">
<hibernate-mapping>
    <class name="edu.db111mil.ventas.dao.Telefono" table="E01_TELEFONO">
        <composite-id name="numero">
            <key-property name="codigoArea" column="codigo_area"/>
            <key-property name="nroTelefono" column="nro_telefono"/>
        </composite-id>
        <property name="tipo"/>
        <many-to-one name="cliente" class="edu.db111mil.ventas.dao.Cliente"
                     column="nro_cliente"/>
    </class>
</hibernate-mapping>
```

Hibernate: Clase Cliente

```
public class Cliente {
    private int nroCliente;
    private String nombre;
    private String apellido;
    private String direccion;
    private List<Telefono> telefono = new ArrayList<>();
    private boolean activo;
    public Cliente(){
        super();
    }
    ...
    public int getNroCliente() {
        return nroCliente;
    }

    public void setNroCliente(int nroCliente) {
        this.nroCliente = nroCliente;
    }

    public String getNombre() {
        return nombre;
    }
    ...
}
```

Hibernate: cliente.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping">
<hibernate-mapping>
    <class name="edu.db111mil.ventas.dao.Cliente" table="E01_CLIENTE">
        <id name="nroCliente" column="nro_cliente"/>
        <property name="nombre"/>
        <property name="apellido"/>
        <property name="direccion"/>
        <property name="activo"/>
        <bag name="telefono" table="E01_TELEFONO" inverse="false" fetch="select">
            <key column="nro_cliente"/>
            <one-to-many class="edu.db111mil.ventas.dao.Telefono"/>
        </bag>
    </class>
</hibernate-mapping>
```

Hibernate: Clase Factura

```
public class Factura implements Serializable{
    private int nroFactura;
    private transient Date fecha;
    private transient double totalSinIva;
    private transient double iva;
    private transient double totalConIva;
    private transient Cliente cliente;
    private transient List<DetalleFactura> detalleFactura;

    public Factura() {
    }
    public Factura(int nroFactura, Date fecha, double totalSinIva, double iva, double totalConIva,
Cliente cliente,
        List<DetalleFactura> detalleFactura) {
        ...
    }
    public int getNroFactura() {
        return nroFactura;
    }
    public void setNroFactura(int nroFactura) {
        this.nroFactura = nroFactura;
    }
    ...
}
```

Hibernate: factura.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping">
<hibernate-mapping>
    <class name="edu.db111mil.ventas.dao.Factura" table="E01_FACTURA">
        <id name="nroFactura" column="nro_factura"/>
        <property name="fecha"/>
        <property name="totalSinIva" column="total_sin_iva"/>
        <property name="iva"/>
        <property name="totalConIva" column="total_con_iva"/>
        <many-to-one name="cliente" class="edu.db111mil.ventas.dao.Cliente"
                     column="nro_cliente"/>

        <bag name="detalleFactura" table="E01_DETALLE_FACTURA" inverse="false" fetch="select">
            <key column="nro_factura"/>
            <one-to-many class="edu.db111mil.ventas.dao.DetalleFactura"/>
        </bag>
    </class>
</hibernate-mapping>
```

Hibernate: Clase DetalleFactura

```
public class DetalleFactura implements Serializable{
    private Factura factura;
    private Producto producto;
    private transient int nroItem;
    private transient int cantidad;

    public DetalleFactura() {
    }

    public DetalleFactura(Factura factura, Producto producto, int nroItem, int cantidad) {
        ...
    }

    public Factura getFactura() {
        return factura;
    }

    public void setFactura(Factura factura) {
        this.factura = factura;
    }

    public Producto getProducto() {
        return producto;
    }
    ...
}
```

Hibernate: detalleFactura.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="edu.db111mil.ventas.dao.DetalleFactura" table="E01_DETALLE_FACTURA">
      <composite-id>
          <key-many-to-one name='factura'
                           class="edu.db111mil.ventas.dao.Factura"
                           column="nro_factura"/>
          <key-many-to-one name='producto'
                           class="edu.db111mil.ventas.dao.Producto"
                           column="codigo_producto"/>
      </composite-id>
      <property name="nroItem" column="nro_item"/>
      <property name="cantidad"/>
  </class>
</hibernate-mapping>
```

Hibernate: Clase Producto

```
public class Producto  implements Serializable{
    private int codigoProducto;
    private transient String marca;
    private transient String nombre;
    private transient String descripcion;
    private transient float precio;
    private transient int stock;

    public Producto() {
    }

    public Producto(int codigoProducto, String marca, String nombre, String descripcion,...) {
        ...
    }

    public int getCodigoProducto() {
        return codigoProducto;
    }

    public void setCodigoProducto(int codigoProducto) {
        this.codigoProducto = codigoProducto;
    }
    ...
}
```

Hibernate: producto.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="edu.db111mil.ventas.dao.Producto" table="E01_PRODUCTO">
      <id name="codigoProducto" column="codigo_producto"/>
      <property name="marca"/>
      <property name="nombre"/>
      <property name="descripcion"/>
      <property name="precio"/>
      <property name="stock"/>
  </class>
</hibernate-mapping>
```

Hibernate: producto.hbm.xml v2

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="edu.db111mil.ventas.dao.Producto" table="E01_PRODUCTO">
      <id name="codigoProducto" column="codigo_producto">
          <generator class="increment"></generator>
      </id>
      <property name="marca"/>
      <property name="nombre"/>
      <property name="descripcion"/>
      <property name="precio"/>
      <property name="stock"/>
  </class>
</hibernate-mapping>
```

Hibernate: Acceso a la base de datos

Hasta el momento solo hemos definido como se mapean las tablas a las clases. Ahora veremos:

- Listar todas las tuplas de una tabla como una lista de instancia de clases.
- Utilizar el lenguaje de consulta de Hibernate. Hibernate Query Language (HQL) que es similar a SQL.
- Buscar una una tupla por su clave primaria.
- Realizar una consulta por una condición.
- Realizar una inserción en la base de datos.
- Realizar una modificación en la base de datos.
- Realizar una eliminación en la base de datos.

Hibernate: Listar tuplas

```
session.beginTransaction();
CriteriaQuery<Cliente> q = session.getCriteriaBuilder().createQuery(Cliente.class);
q.select(q.from(Cliente.class));
List<Cliente> l = session.createQuery(q).list();
System.out.println("Lista de clientes");
for(Cliente cl: l){
    System.out.println(cl);
}
session.getTransaction().commit();
```

Hibernate: Listar tuplas

```
session.beginTransaction();
Query q = session.createQuery("from Cliente", Cliente.class);
List<Cliente> l = q.list();
System.out.println("Lista de clientes no activos");
for(Cliente cl: l){
    System.out.println(cl);
}
session.getTransaction().commit();
```

Hibernate: Buscar una tupla por PK

```
int i = //id a buscar;
session.beginTransaction();
System.out.println("Buscando Factura "+i);
Factura f = session.get(Factura.class, i);
System.out.println(f);
session.getTransaction().commit();
```

Hibernate: Listar tuplas que cumplan una condición

```
session.beginTransaction();
Query q = session.createQuery("from Cliente where activo = :act", Cliente.class);
q.setParameter("act", false);
List<Cliente> l = q.list();
System.out.println("Lista de clientes no activos");
for(Cliente cl: l){
    System.out.println(cl);
}
session.getTransaction().commit();
```

Hibernate: Insertar un nuevo objeto/tupla

```
int i = //Nuevo id;
session.beginTransaction();
System.out.println("Agregando cliente "+i);
Cliente c = new Cliente(i, "Agregado", "Por", "Hibernate", new ArrayList<Telefono>(),
        true);
session.save(c);
session.getTransaction().commit();
```

Hibernate: Modificar objetos/tuplas

```
int i = //id a modificar;
session.beginTransaction();
System.out.println("Modificando cliente "+i);
Cliente c = session.get(Cliente.class, i);
c.setNombre("Modificado");
session.getTransaction().commit();
```

Hibernate: Modificar objetos/tuplas HQL

```
int i = //id a modificar;
session.beginTransaction();
System.out.println("Modificando cliente "+i);
Query query = session.createQuery("update Clente set nombre = :idCliente" +
                                  " where nroCliente = :nuevoNombre");
query.setParameter("nroCliente", i);
query.setParameter("nuevoNombre", "Modificado");
int result = query.executeUpdate();
session.getTransaction().commit();
```

Hibernate: Borrar objetos/tuplas

```
int i = //id a borrar;
session.beginTransaction();
System.out.println("Borrando cliente "+i);
Cliente c = session.get(Cliente.class, i);
session.delete(c);
session.getTransaction().commit();
```

Hibernate: Borrar objetos/tuplas

```
int i = //id a borrar;
session.beginTransaction();
System.out.println("Borrando cliente "+i);
Query query = session.createQuery("delete Cliente where nroCliente = :idCliente");
query.setParameter("idCliente", "7277");
int result = query.executeUpdate();
session.getTransaction().commit();
```

Hibernate: Queries complejas

```
session.beginTransaction();
Query q = session.createQuery("select max(nroCliente) from Cliente");
List<?> l = q.list();
System.out.println("Maximo nroCliente: "+l.get(0));
session.getTransaction().commit();
```

Para más información de HQL, JPQL se recomienda leer:
https://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html