	UNIVERSIDAD DON BOSCO **FACULTA DE INGENIERIA** **ESCUELA DE COMPUTACION**
Ciclo II	**Desarrollo de aplicaciones con Web Frameworks** **Guía de Laboratorio No. 6** **Introducción a Hibernate**

I. OBJETIVOS.

Que el alumno

- Implemente el framework de hibernate para el acceso a base de datos.
- Implementar Struts e Hibernate en un mismo proyecto.

II. INTRODUCCION

1. Capas de persistencia y posibilidades que estas ofrecen.

En el diseño de una aplicación una parte muy importante es la manera en la cual accedemos a nuestros datos en la base de datos. Determinar esta parte se convierte en un punto crítico para el futuro desarrollo.

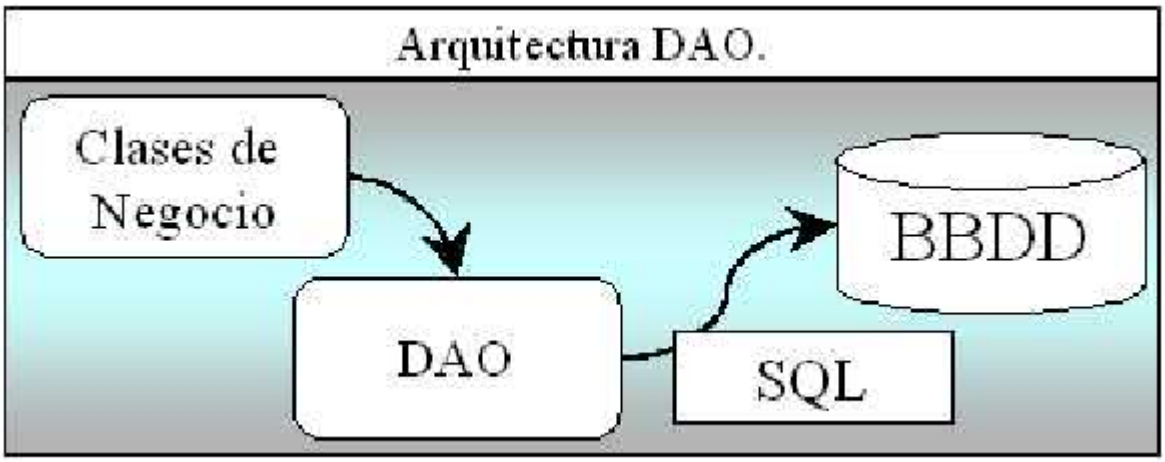
La manera tradicional de acceder es a través de JDBC directamente conectado a la BBDD mediante ejecuciones de sentencias SQL:

1. Sentencias SQL directas

Esta primera aproximación puede ser útil para proyectos o arquitecturas con pocas clases de negocio, sin embargo, debido a que el mantenimiento del código está altamente ligado a los cambios en el modelo de datos relacional de la BBDD, un mínimo cambio implica la revisión de casi todo el código, así como su compilación y nueva instalación en el cliente.

El acceso a través de SQL directas puede ser utilizado de manera puntual para realizar operaciones a través del lenguaje SQL lo cual sería mucho más efectivo que la carga de gran cantidad de objetos en memoria. Si bien un buen motor de persistencia debería implementar mecanismos para ejecutar estas operaciones masivas sin necesidad de acceder a este nivel.

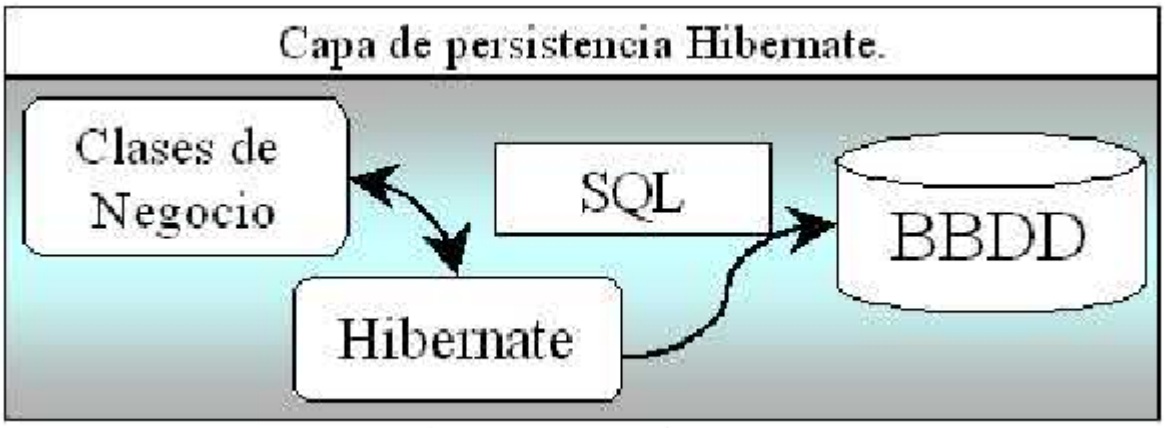
Una aproximación más avanzada seria la creación de unas clases de acceso a datos (DAO Data Acces Object). De esta manera nuestra capa de negocio interactuaría con la capa DAO y esta sería la encargada de realizar las operaciones sobre la BBDD.



2. Ejemplo de DAO (Data Access Object

Los problemas de esta implementación siguen siendo el mantenimiento de la misma así como su portabilidad. Lo único que podemos decir es que tenemos el código de transacciones encapsulado en las clases DAO. Un ejemplo de esta arquitectura podría ser Microsoft ActiveX Data Object (ADO).

¿Y cómo encaja Hibernate en todo esto? Lo que parece claro es que debemos separar el código de nuestras clases de negocio de la realización de nuestras sentencias SQL contra la BBDD. Por lo tanto, Hibernate es el puente entre nuestra aplicación y la BBDD, sus funciones van desde la ejecución de sentencias SQL a través de JDBC hasta la creación, modificación y eliminación de objetos persistentes.



3. Persistencia con Hibernate

Con la creación de la capa de persistencia se consigue que los desarrolladores no necesiten conocer nada acerca del esquema utilizado en la BBDD. Tan solo conocerán el interface proporcionado por nuestro motor de persistencia. De esta manera conseguimos separar de manera clara y definida, la lógica de negocios de la aplicación con el diseño de la BBDD.

Esta arquitectura conllevará un proceso de desarrollo más costoso pero una vez se encuentre implementada las ventajas que conlleva merecerán la pena. Es en este punto donde entra en juego Hibernate. Como capa de persistencia desarrollada tan solo tenemos que adaptarla a nuestra arquitectura.

2. Qué es hibernate

Hibernate es una capa de persistencia objeto/relacional y un generador de sentencias sql. Te permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. De una manera muy rápida y optimizada podremos generar BBDD en cualquiera de los entornos soportados: Oracle, DB2, MySql, etc... Y lo más importante de todo, es open source, lo que supone, entre otras cosas, que no tenemos que pagar nada por adquirirlo.

Uno de los posibles procesos de desarrollo consiste en, una vez tengamos el diseño de datos realizado, mapear este a ficheros XML siguiendo la DTD de mapeo de Hibernate. Desde estos podremos generar el código de nuestros objetos persistentes en clases Java y también crear BBDD independientemente del entorno escogido.

Hibernate se integra en cualquier tipo de aplicación justo por encima del contenedor de datos. Una posible configuración básica de hibernate es la siguiente:

4. Arquitectura Base

Conceptos básicos

Hibernate funciona asociando a cada tabla de la base de datos un Plain Old Java Object (POJO, a veces llamado Plain Ordinary Java Object). Un POJO es similar a una Java Bean, con propiedades accesibles mediante métodos setter y getter, como, por ejemplo:

```
package net.sf.hibernate.examples.quickstart;

public class Cat {

    private String id;
    private String name;
    private char sex;
```

```
    private float weight;

    public Cat() {
    }

    public String getId() {
        return id;
    }

    private void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public char getSex() {
        return sex;
    }

    public void setSex(char sex) {
        this.sex = sex;
    }

    public float getWeight() {
        return weight;
    }

    public void setWeight(float weight) {
        this.weight = weight;
    }

}
```

Para poder asociar el POJO a su tabla correspondiente en la base de datos, Hibernate usa los ficheros hbm.xml.

Para la clase Cat se usa el fichero Cat.hbm.xml para mapearlo con la base de datos. En este fichero se declaran las propiedades del POJO y sus correspondientes nombres de columna en la base de datos, asociación de tipos de datos, referencias, relaciones x a x con otras tablas etc:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping
    PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping>

    <class name="net.sf.hibernate.examples.quickstart.Cat" table="CAT">

        <!-- A 32 hex character is our surrogate key. It's automatically
            generated by Hibernate with the UUID pattern. -->
        <id name="id" type="string" unsaved-value="null" >
            <column name="CAT_ID" sql-type="char(32)" not-null="true"/>
            <generator class="uuid.hex"/>
        </id>

        <!-- A cat has to have a name, but it shouldn' be too long. -->
        <property name="name">
            <column name="NAME" length="16" not-null="true"/>
        </property>

        <property name="sex"/>

        <property name="weight"/>

    </class>

</hibernate-mapping>
```

De esta forma en nuestra aplicación podemos usar el siguiente código para comunicarnos con nuestra base de datos:

```
sessionFactory = new Configuration().configure().buildSessionFactory();
Session session = sessionFactory.openSession();

Transaction tx= session.beginTransaction();

Cat princess = new Cat();
princess.setName("Princess");
princess.setSex('F');
princess.setWeight(7.4f);

session.save(princess);
tx.commit();

session.close();
```

Además, tiene la ventaja de que nos es totalmente transparente el uso de la base de datos pudiendo cambiar de base de datos sin necesidad de cambiar una línea de código de nuestra aplicación, simplemente cambiando los ficheros de configuración de Hibernate.

III. PROCEDIMIENTO

Para esta guía utilizaremos Netbeans. Su docente le explicará las herramientas que utilizará para consultas y pruebas.

Pasos Previos

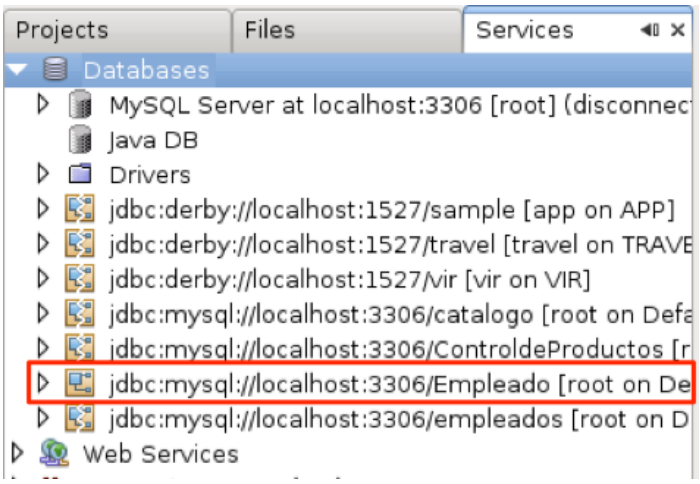
Paso 1

Crear una base de datos llamada **“Empleado”** y una tabla llamada **“datosempleados”** con la siguiente estructura

Campo	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
Id	int(11)			No	*None*	auto_increment
Nombres	varchar(80)	latin1_swedish_ci		No	*None*	
Apellidos	varchar(80)	latin1_swedish_ci		No	*None*	
Edad	int(11)			No	*None*	
Telefono	varchar(10)	latin1_swedish_ci		Sí	*NULL*	
Direccion	varchar(9)	latin1_swedish_ci		Sí	*NULL*	

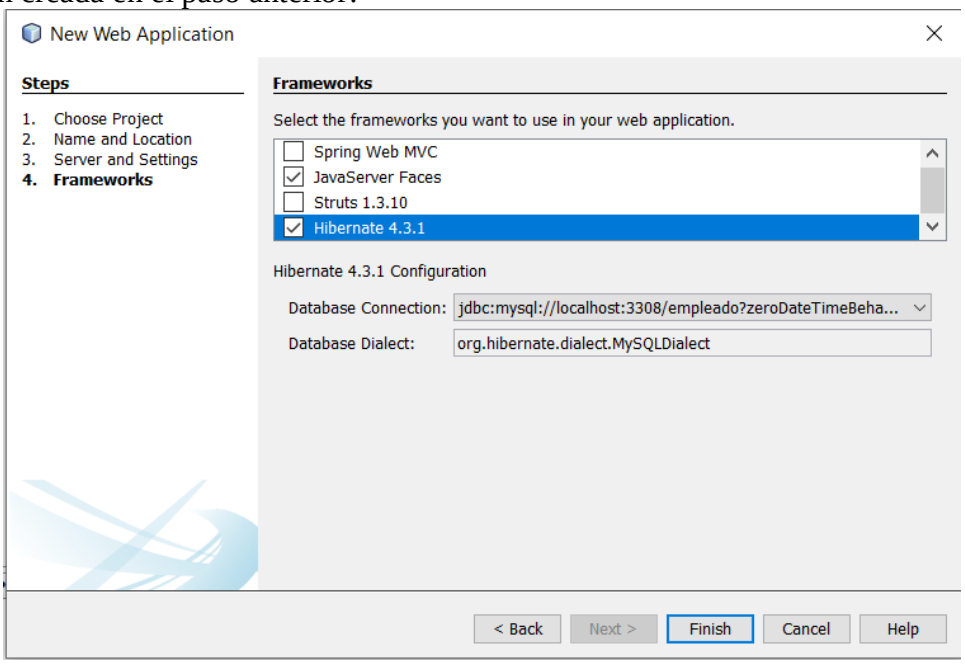
Paso 2

Crear una nueva conexión en netbeans, para esto ir a la pestaña **Service** seleccionar **"New Connection"** y agregar los parámetros necesarios para establecer la conexión a la base de datos **Empleado**:



Paso 3

Crear un nuevo proyecto llamado **Hibernate,** seleccionar como framework de implementación **"JSF e Hibernate",** en la pestaña **Database Connection** implementar la conexión creada en el paso anterior:



Los pasos realizados hasta este punto permiten la creación de un proyecto con soporte para el framework de **"JSF e Hibernate"**. Es importante conocer los archivos de Hibernate entre estos está el archivo **"hibernate.cfg.xml"** que se ha creado a la hora de seleccionar el framework, el archivo tiene una estructura similar a la siguiente.

```
<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/Empleado</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">rafael</property>
  </session-factory>
</hibernate-configuration>
```

Nota: Verifique los parámetros de configuración e interprételos, su docente le explicará si tiene dudas.

Implementando Hibernate

Para esta parte crearemos los POJO y el mapeo correspondiente de estos con las tablas de la base de datos, para hacer esto más fácil utilizaremos el asistente de netbeans para la creación de estos archivos.

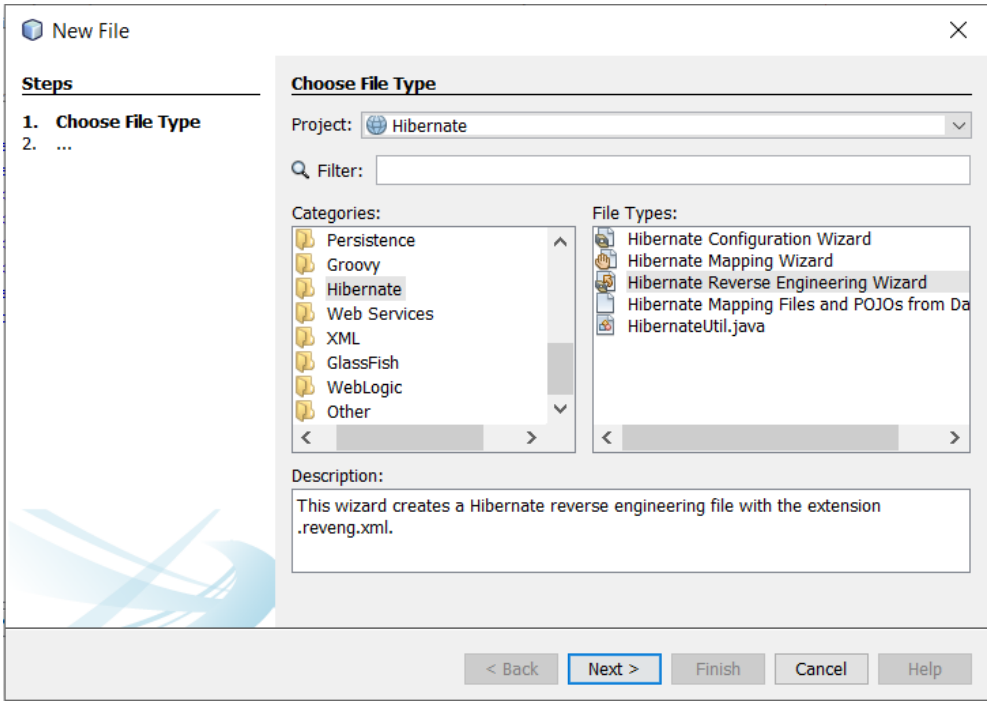
Paso1

Creación de archivos de Mapeo y clases de Java

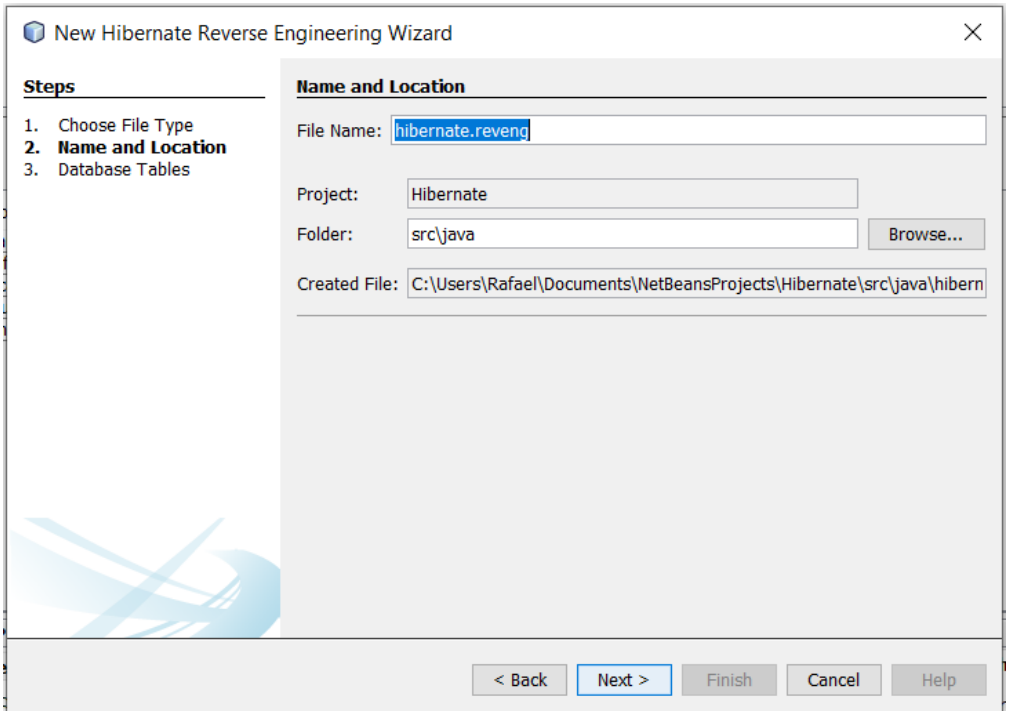
Lo siguiente será crear los archivos de mapeo y las clases de java a partir de nuestra base de datos. Al generar esto, Netbeans lo que hará será tratar de relacionar la base de datos con clases de java (Entidades), creando por cada tabla una clase java con sus atributos (filas de la tabla) y un archivo XML en donde se especifican características de "objeto-tabla" como por ejemplo el nombre de la base de datos (catalog), el nombre de la tabla (table) y otras varias características más.

Empezaremos por crear un archivo de ingeniería inversa (hibérnate.reveng.xml) archivo que guardara la configuración de nuestras tablas, para eso realizamos los siguientes pasos:

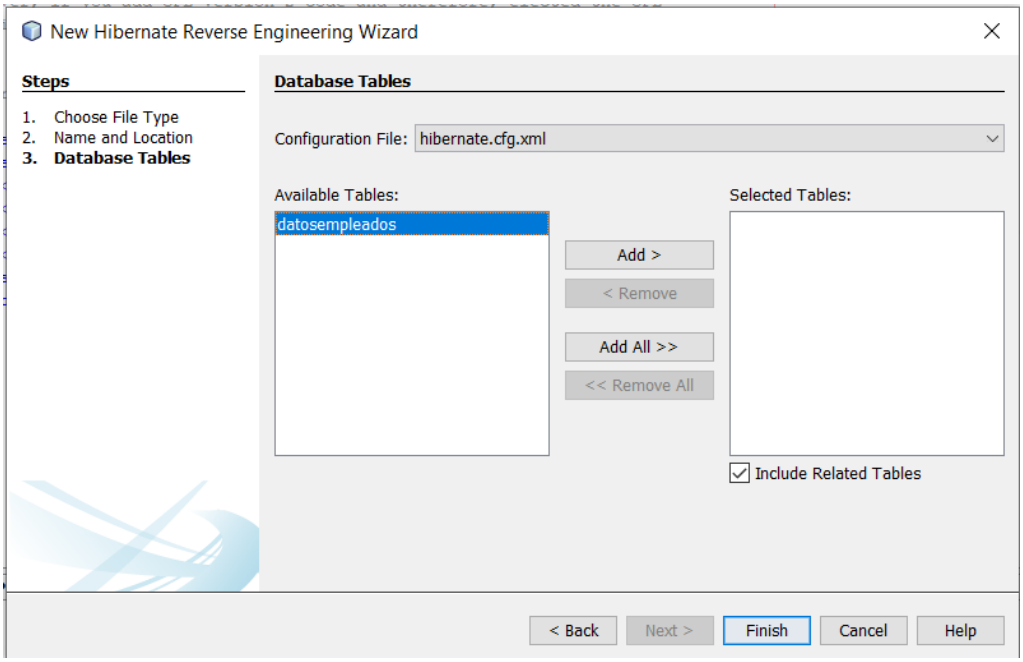
File-> New File-> Categories / Hibernate > Tipo de Archivo / Asistente de ingeniería inversa de Hibernate. El asistente nos mostrará lo siguiente:



Le dejamos el nombre por default y la ruta será **src/java** esto es el lugar donde guardará el archivo junto con el archivo **hibernate.cfg.xml** en el paquete predeterminado. Clic en siguiente:

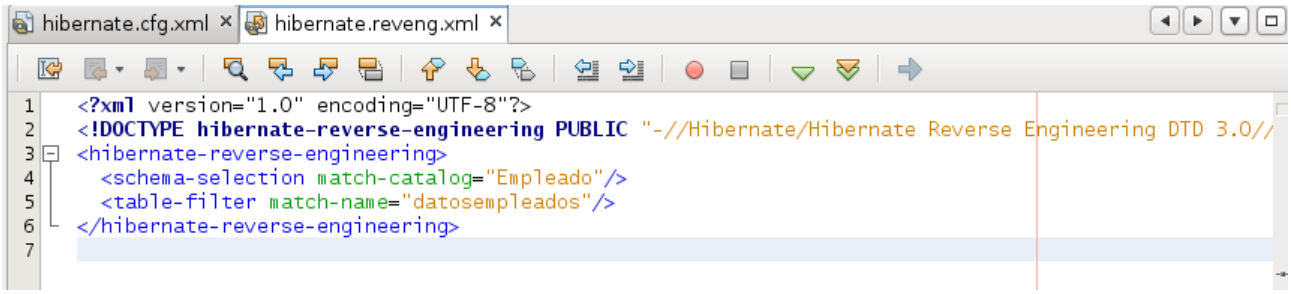


Seleccionamos nuestro archivo de configuración y agregamos las tablas a utilizar, para este caso agregar la tabla **“datosempleados”**.



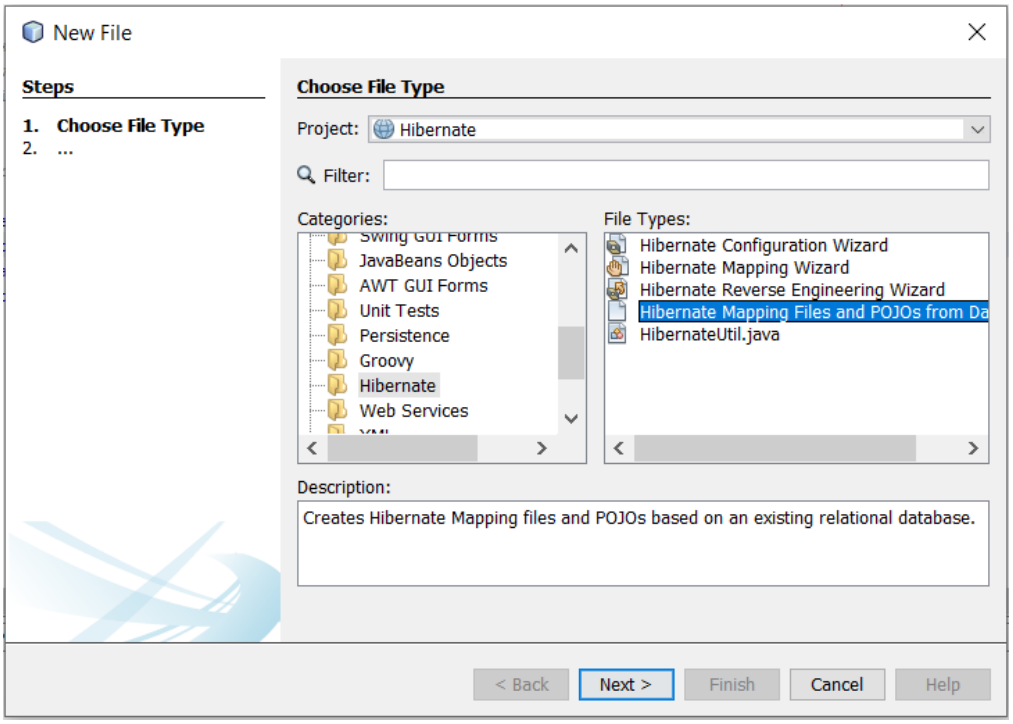
Click en **Finish**.

Ahora vamos al paquete predeterminado y vemos como junto al archivo **hibernate.cfg.xml** quedo nuestro **hibernate.reveng.xml**, doble clic sobre el icono y veremos su código correspondiente.

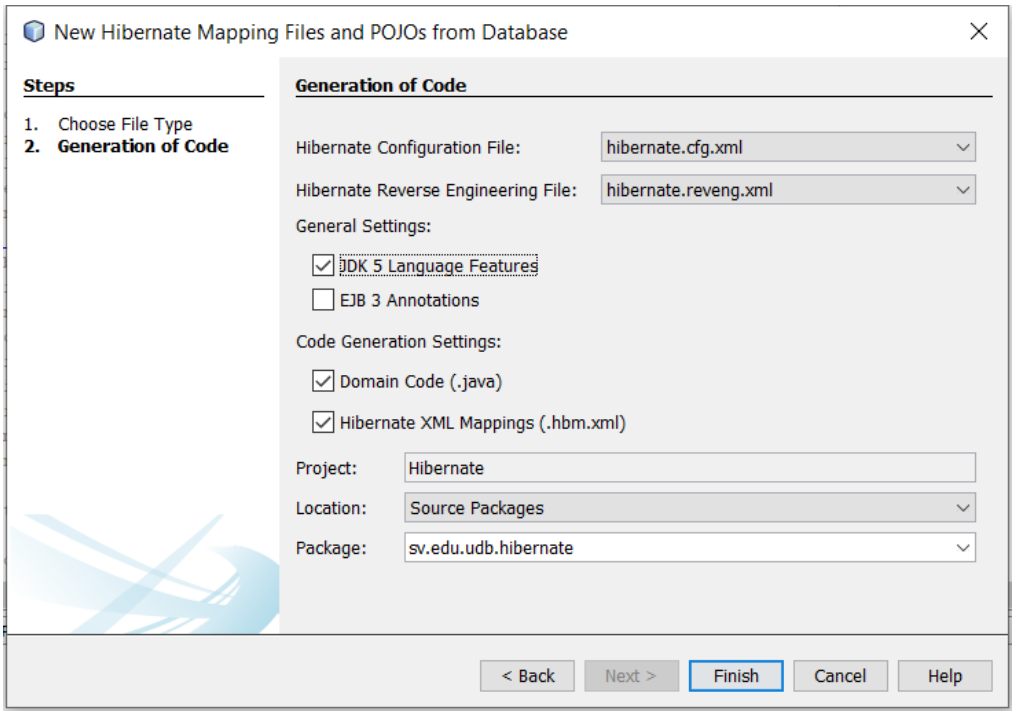


Paso 2

Luego de creado nuestro archivo de configuración de tablas procederemos a crear los Archivos de Mapeo de Hibernate, para esto crear un nuevo paquete llamado **"sv.edu.udb.hibernate"** dar click derecho, seleccionar other buscar dentro de la categoría Hibernate y seleccionar **"Hibarnate Mapping Files and POJOS from Database"**.



Seleccionamos si no lo están nuestros archivos de configuración. Marcar la casilla de características del lenguaje JDK 5 y lo demás lo dejamos como viene por defecto, también. debemos de indicar en que package para nuestro caso **"sv.edu.udb.hibernate"**.

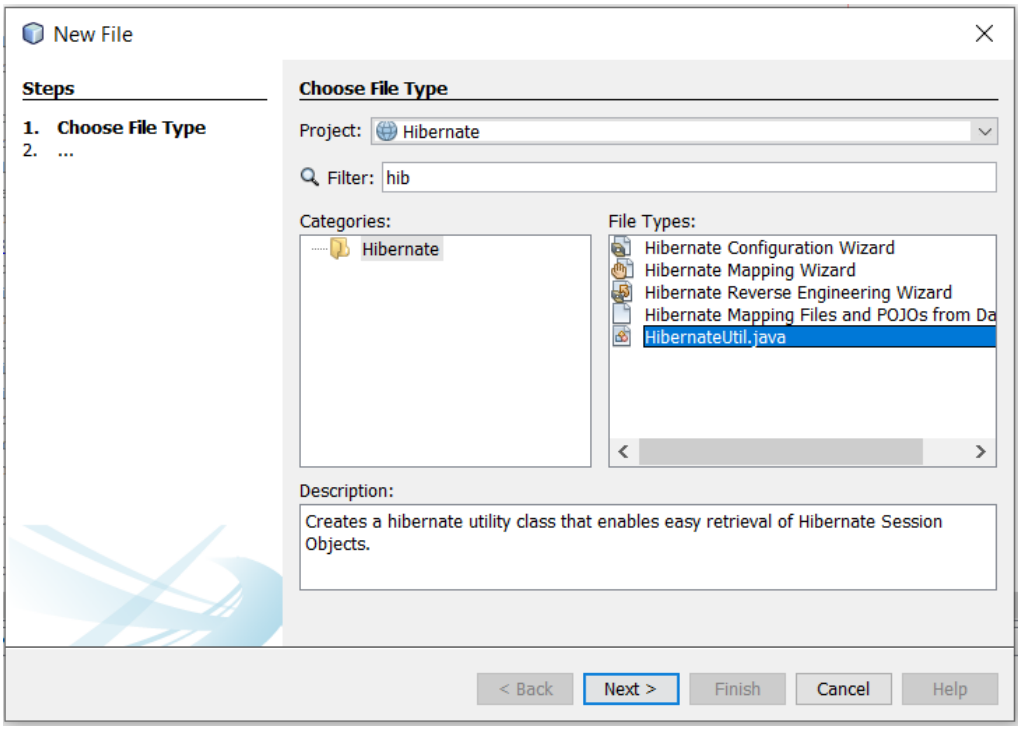


Dar click en **Finish**.

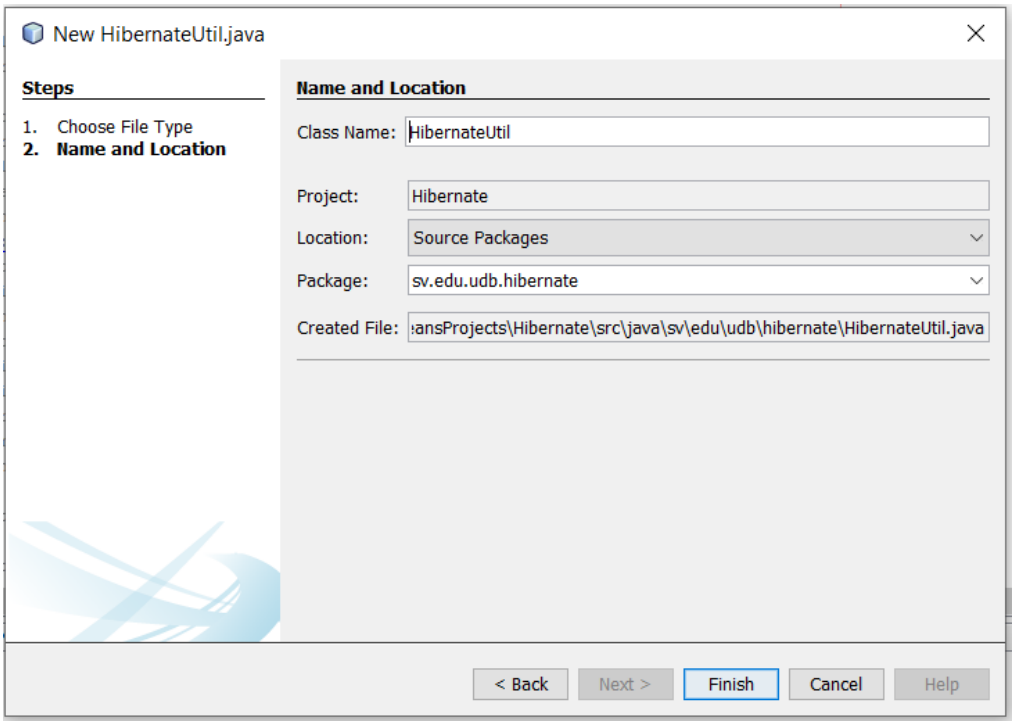
Al observar el paquete podemos ver los nuevos archivos el **POJO** y al archivo de mapeo.
Nota: estos archivos serán explicados por su docente.

Paso 3
El siguiente paso será crear el archivo HibernateUtil.java. Este archivo lo que hace es crear

un SessionFactory (objeto) que cargará la información sobre dónde se encuentran los ficheros de mapeo de Hibernate. Dar click derecho sobre el paquete **"sv.edu.udb.hibernate"** seleccionar other, buscar dentro de la categoría Hibernate y seleccionar **"Hibarnate Mapping Files and POJOS from Database"**...



Después de dar clic en siguiente en la pantalla anterior aparecerá una nueva ventana en la cual debemos dar el nombre a este archivo, para nuestro caso poner el nombre **"HibernateUtil"**.



Dar click en **Finish**

Sobrescribir la clase **HibernateUtil** con el siguiente código:

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package sv.edu.udb.hibernate;

import org.hibernate.SessionFactory;

import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.boot.registry.StandardServiceRegistry;

/**
 * @author Rafael Torres
 *
 */
public class HibernateUtil {
   private static final SessionFactory sessionFactory;

   static {
        try {
                Configuration cfg = new Configuration().configure("hibernate.cfg.xml");

                StandardServiceRegistryBuilder sb = new
StandardServiceRegistryBuilder();
                sb.applySettings(cfg.getProperties());
                StandardServiceRegistry standardServiceRegistry = sb.build();
                sessionFactory = cfg.buildSessionFactory(standardServiceRegistry);
        } catch (Throwable th) {
             System.err.println("Enitial SessionFactory creation failed" + th);
             throw new ExceptionInInitializerError(th);
        }
   }
   public static SessionFactory getSessionFactory() {
        return sessionFactory;
   }
}
```

Hasta este punto ya hemos creado los POJO y el mapeo que necesitamos, podemos hacer unas pruebas con una simple clase, para esto crear dentro del paquete **"sv.edu.udb.hibernate",** una clase que contenga un main, esta clase será llamada **"Test"**.

Para ingresar un nuevo registro, agregar el siguiente código dentro de la main

```
SessionFactory sesFact=HibernateUtil.getSessionFactory();
Session ses=sesFact.openSession();
```

```
        Transaction tra=ses.beginTransaction();

        Datosempleados datos=new Datosempleados();
        datos.setNombres("Laura");
        datos.setApellidos("Ramirez");
        datos.setEdad(23);
        datos.setTelefono("22202222");
        datos.setDireccion("Mi casa");

        ses.save(datos);
        tra.commit();
```

Como se puede ver en el código no hay ninguna sentencia Sql para el ingreso de datos, es aquí una de las ventajas de hibernate.

Agregar los siguientes import que son necesarios para la implementación de hibernate

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
```

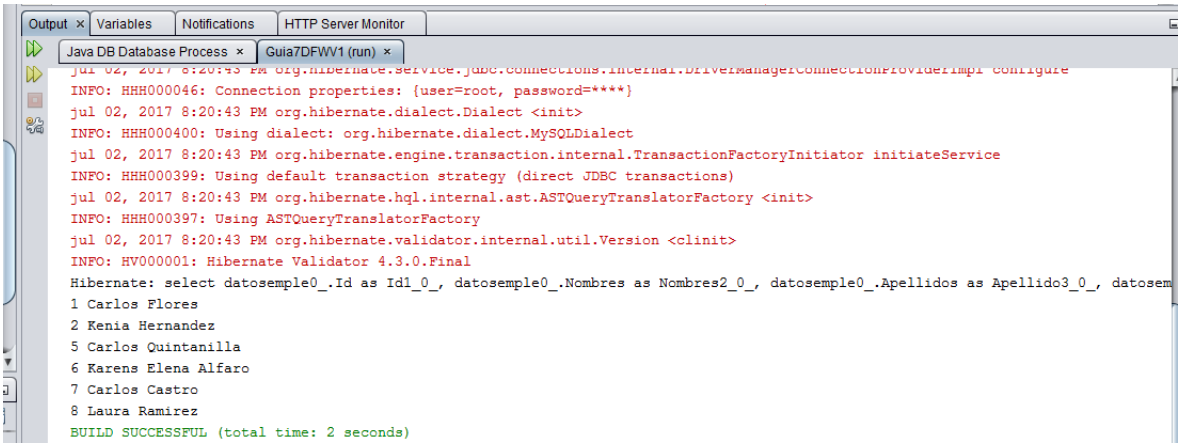
Proceder a correr el archivo **Test.java** y observar en la base de datos que el registro ha sido ingresado.

Proceder a Crear una nueva clase llamada **TestResult.java** y agregar el siguiente código, que permitirá ver el resultado en la base de datos, utilizando consulta HQL.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package sv.edu.udb.hibernate;

/**
 *
 * @author Rafael Torres
 */

import java.util.ArrayList;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
public class TestResult {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
```

```
        SessionFactory sesFact=HibernateUtil.getSessionFactory();
        Session ses=sesFact.openSession();
        ArrayList<Datosempleados> listaEmpleado=new ArrayList<Datosempleados>();
        String sql="from Datosempleados";
        listaEmpleado= (ArrayList<Datosempleados>) ses.createQuery(sql).list();
        for(int i=0;i<listaEmpleado.size();i++){
        Datosempleados empleado=(Datosempleados)listaEmpleado.get(i);
        System.out.println(empleado.getId() + " " + empleado.getNombres() + " " +
empleado.getApellidos() );

      }
   }

}
```

Resultado



Si queremos agregar un Update o un delete ahora necesitamos agregar el id para saber que registro actualizaremos o eliminaremos.

Para la operación “actualizar” debemos modificar el código como se muestra a continuación:

```
    SessionFactory sesFact=HibernateUtil.getSessionFactory();
    Session ses=sesFact.openSession();
    Transaction tra=ses.beginTransaction();

    Datosempleados datos=new Datosempleados();
    datos.setId(6);
    datos.setNombres("Laura Carolina");
    datos.setApellidos("Ramirez");
    datos.setEdad(23);
    datos.setTelefono("22202222");
    datos.setDireccion("Mi casa2");

    ses.update(datos);
    tra.commit();
```

Para eliminar registros debemos cambiar el método update por delete.

INTEGRACIÓN CON JAVA SERVER FACES

Paso 1.
Crear una nueva clase llamada "**DatosEmpleadosDAO**" en el paquete "**sv.edu.udb.dao**" y agregar el siguiente código.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package sv.edu.udb.dao;

import java.util.List;
import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import sv.edu.udb.hibernate.Datosempleados;
import sv.edu.udb.hibernate.HibernateUtil;

/**
 *
 * @author Rafael Torres
 */
public class DatosEmpleadosDAO {

   public void addEmpleado(Datosempleados empleados){
       SessionFactory sesFact = HibernateUtil.getSessionFactory();
       Session ses = sesFact.openSession();
       Transaction tra = null;

      try {
        tra=ses.beginTransaction();
        ses.save(empleados);
        ses.getTransaction().commit();
      } catch (Exception e) {
         e.printStackTrace();
         if ( tra != null ){
            tra.rollback();
         }
      }finally{
         ses.flush();
         ses.close();
      }
   }
   public void deleteEmpleado(Integer idEmpleado){
```

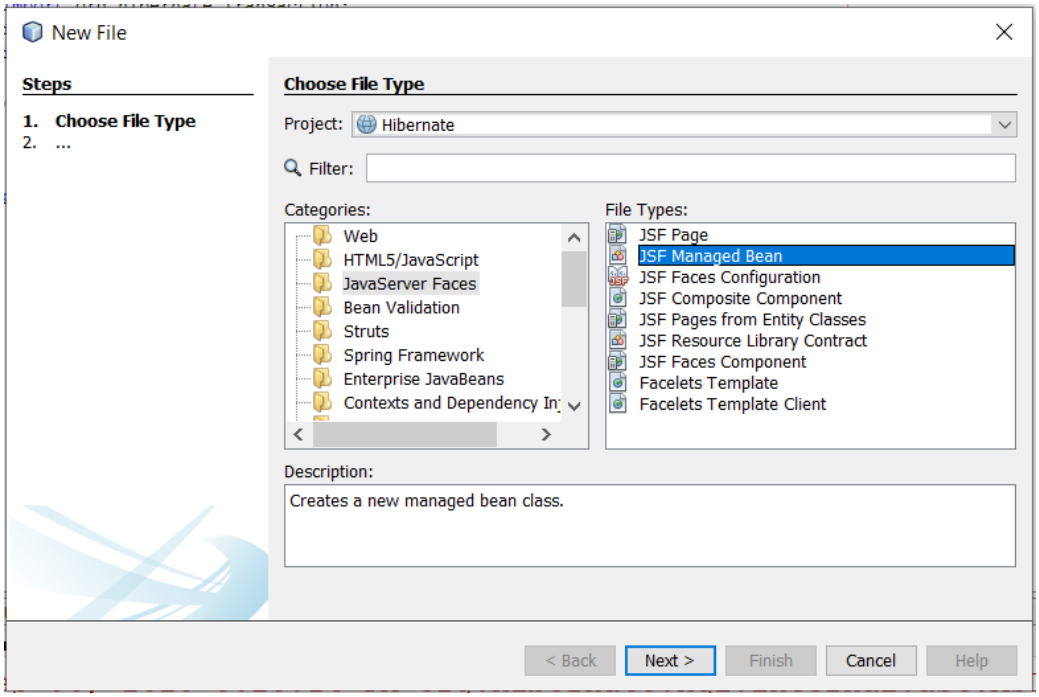
```
        SessionFactory sesFact = HibernateUtil.getSessionFactory();
        Session ses = sesFact.openSession();
        Transaction tra = null;

       try {
         tra=ses.beginTransaction();
         Datosempleados empleado = (Datosempleados)
ses.get(Datosempleados.class,idEmpleado);
         ses.delete(empleado);
         ses.getTransaction().commit();
       } catch (Exception e) {
          e.printStackTrace();
          if ( tra != null ){
             tra.rollback();
          }
       }finally{
          ses.flush();
          ses.close();
       }

   }
   public void updateEmpleado(Integer idEmpleado, Datosempleados newEmpleados){
        SessionFactory sesFact = HibernateUtil.getSessionFactory();
        Session ses = sesFact.openSession();
        Transaction tra = null;

       try {
         tra=ses.beginTransaction();
         Datosempleados empleado = (Datosempleados)
ses.load(Datosempleados.class,idEmpleado);
         empleado.setId(empleado.getId());
         empleado.setNombres(newEmpleados.getNombres());
         empleado.setApellidos(newEmpleados.getApellidos());
         empleado.setEdad(newEmpleados.getEdad());
         empleado.setDireccion(newEmpleados.getDireccion());

         ses.update(empleado);
         ses.getTransaction().commit();
       } catch (Exception e) {
          e.printStackTrace();
          if ( tra != null ){
             tra.rollback();
          }
       }finally{
          ses.flush();
          ses.close();
       }
   }

   public Datosempleados getEmpleadoID(Integer idEmpleado){
```

```
        Datosempleados empleado = null;
        SessionFactory sesFact = HibernateUtil.getSessionFactory();
        Session ses = sesFact.openSession();
        Transaction tra = null;

        try {
          tra=ses.beginTransaction();
          String queryString = "from Datosempleados where id = :idFind";
          Query query = ses.createQuery(queryString);
          query.setParameter("idFind", idEmpleado);
          empleado = (Datosempleados) query.uniqueResult();
        } catch (HibernateException e) {
           e.printStackTrace();
           if ( tra != null ){
              tra.rollback();
           }
        }finally{
           ses.flush();
           ses.close();
        }

        return empleado;
    }

    public List<Datosempleados> obtenerEmpleados(){
        List<Datosempleados> empleados = null;
        SessionFactory sesFact = HibernateUtil.getSessionFactory();
        Session ses = sesFact.openSession();
        Transaction tra = null;

        try {
          tra=ses.beginTransaction();
          String queryString = "from Datosempleados";
          Query query = ses.createQuery(queryString);
          empleados= query.list();

        } catch (HibernateException e) {
           e.printStackTrace();
           if ( tra != null ){
              tra.rollback();
           }
        }finally{
           ses.flush();
           ses.close();
        }
        return empleados;
    }

}
```

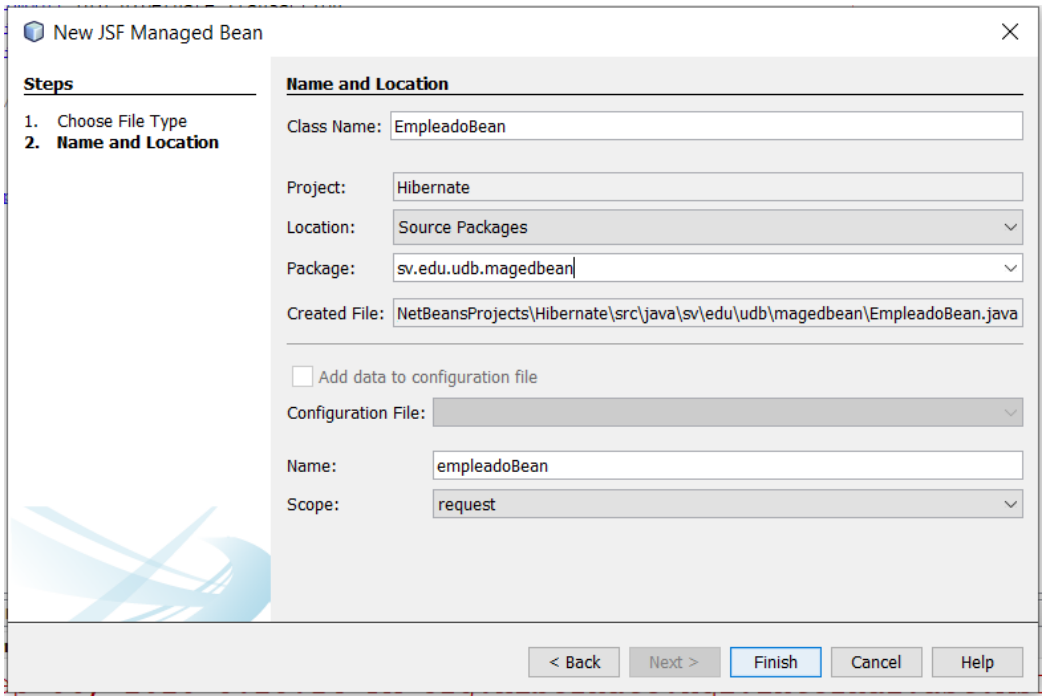
Paso 2.

Crear un **ManagedBean** llamado "**EmpleadoBean**" en el paquete "**sv.edu.udb.magedbean**", ver la siguiente imagen.

- Seleccionar opción **"JSF Manaed Bean"**



- Ingresar el nombre "**EmpleadoBean**" en el paquete "**sv.edu.udb.managebean**"



- Agregar el siguiente código al ManagedBean creado anteriormente.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

package sv.edu.udb.magedbean;

import java.util.List;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;
import sv.edu.udb.dao.DatosEmpleadosDAO;
import sv.edu.udb.hibernate.Datosempleados;

/**
 *
 * @author Rafael Torres
 */
@ManagedBean
@SessionScoped
public class EmpleadoBean {

    private Integer id;
    private String nombres;
    private String apellidos;
    private int edad;
    private String telefono;
    private String direccion;

   /**
    * Creates a new instance of EmpleadoBean
    */
   public EmpleadoBean() {
   }

   public void addEmpleado( ){
      DatosEmpleadosDAO empleadoDao = new DatosEmpleadosDAO();
      Datosempleados empleado = new Datosempleados( nombres,  apellidos,  edad,
telefono,  direccion);
      empleadoDao.addEmpleado(empleado);
   }

   public void returnEmpleadoId(){
      DatosEmpleadosDAO empleadoDao = new DatosEmpleadosDAO();
      Datosempleados empleado = empleadoDao.getEmpleadoID(getId()) ;

      if(empleado != null){
```

```
            setId(empleado.getId());
            setNombres(empleado.getNombres() );
            setApellidos(empleado.getApellidos());
            setEdad(empleado.getEdad());
            setTelefono(empleado.getTelefono());
            setDireccion(empleado.getDireccion());
        }else{
            setId(0);
            setNombres("");
            setApellidos("");
            setEdad(0);
            setTelefono("");
            setDireccion("");
            FacesContext.getCurrentInstance().addMessage(null,
                    new FacesMessage("Cliente NO especificado"));
        }

    }

    public String deleteEmpleado(){
      DatosEmpleadosDAO empleadoDao = new DatosEmpleadosDAO();
      Datosempleados empleado = empleadoDao.getEmpleadoID(getId()) ;

      if(empleado != null){
      empleadoDao.deleteEmpleado(getId());
      setId(getId());
      setNombres("");
      setApellidos("");
      setEdad(0);
      setTelefono("");
      setDireccion("");
      FacesContext.getCurrentInstance().addMessage(null,
                    new FacesMessage("Cliente con ID " + getId() + " Eliminado" ));
      }else{
            FacesContext.getCurrentInstance().addMessage(null,
                    new FacesMessage("Cliente con ID " + getId() + " NO encontrado" ));
      }

      return "Empleado";

    }

     public String updateEmpleado(){
      DatosEmpleadosDAO empleadoDao = new DatosEmpleadosDAO();
      Datosempleados obtempleado = empleadoDao.getEmpleadoID(getId()) ;

      if(obtempleado != null){
            Datosempleados empleado = new Datosempleados( nombres,  apellidos,  edad,
telefono,  direccion);

            empleadoDao.updateEmpleado(getId(), empleado);
```

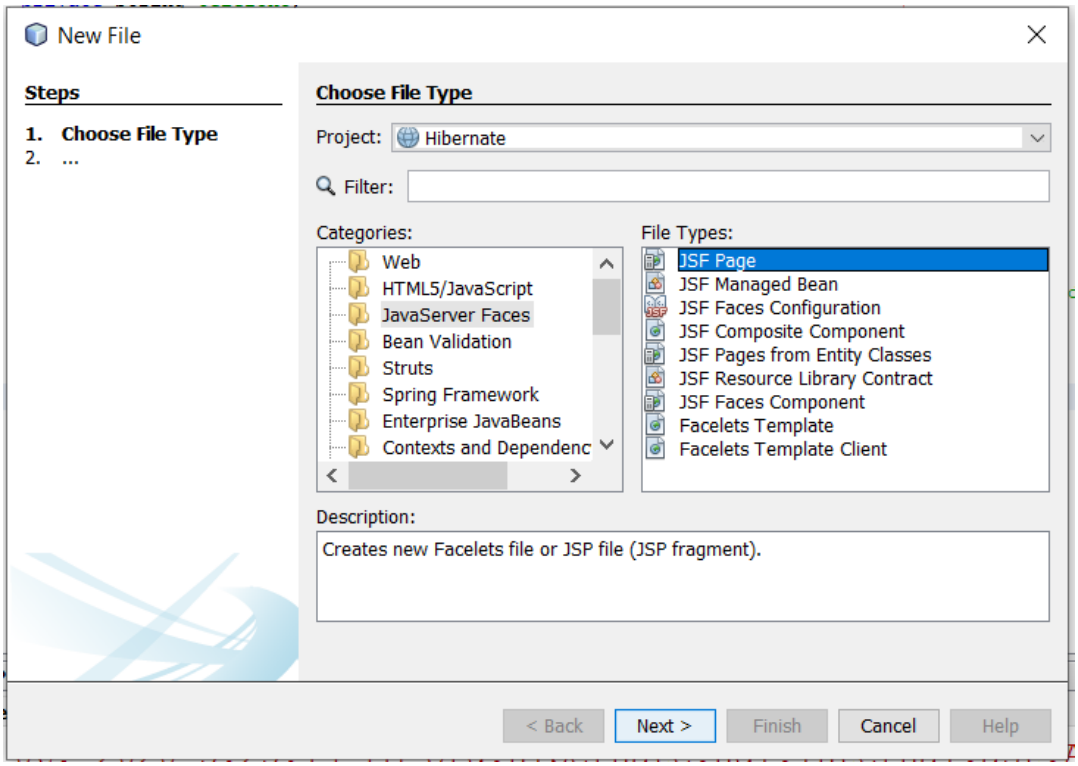
```
        obtempleado = empleadoDao.getEmpleadoID(getId()) ;
        setId(obtempleado.getId());
        setNombres(obtempleado.getNombres() );
        setApellidos(obtempleado.getApellidos());
        setEdad(obtempleado.getEdad());
        setTelefono(obtempleado.getTelefono());
        setDireccion(obtempleado.getDireccion());
        FacesContext.getCurrentInstance().addMessage(null,
                new FacesMessage("Cliente con ID " + getId() + " Actualizado" ));
    }else{
        setId(0);
        setNombres("");
        setApellidos("");
        setEdad(0);
        setTelefono("");
        setDireccion("");
        FacesContext.getCurrentInstance().addMessage(null,
                new FacesMessage("Cliente con ID " + getId() + " NO encontrado" ));
    }

    return "Empleado";

}

public List<Datosempleados> getEmpleados(){
    DatosEmpleadosDAO empleadoDao = new DatosEmpleadosDAO();
    List<Datosempleados> lista = empleadoDao.obtenerEmpleados();
    return lista;
}

/**
 * @return the id
 */
public Integer getId() {
    return id;
}

/**
 * @param id the id to set
 */
public void setId(Integer id) {
    this.id = id;
}

/**
 * @return the nombres
 */
public String getNombres() {
    return nombres;
}
```

```
    /**
     * @param nombres the nombres to set
     */
    public void setNombres(String nombres) {
        this.nombres = nombres;
    }

    /**
     * @return the apellidos
     */
    public String getApellidos() {
        return apellidos;
    }

    /**
     * @param apellidos the apellidos to set
     */
    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }

    /**
     * @return the edad
     */
    public int getEdad() {
        return edad;
    }

    /**
     * @param edad the edad to set
     */
    public void setEdad(int edad) {
        this.edad = edad;
    }

    /**
     * @return the telefono
     */
    public String getTelefono() {
        return telefono;
    }

    /**
     * @param telefono the telefono to set
     */
    public void setTelefono(String telefono) {
        this.telefono = telefono;
    }

    /**
     * @return the direccion
```

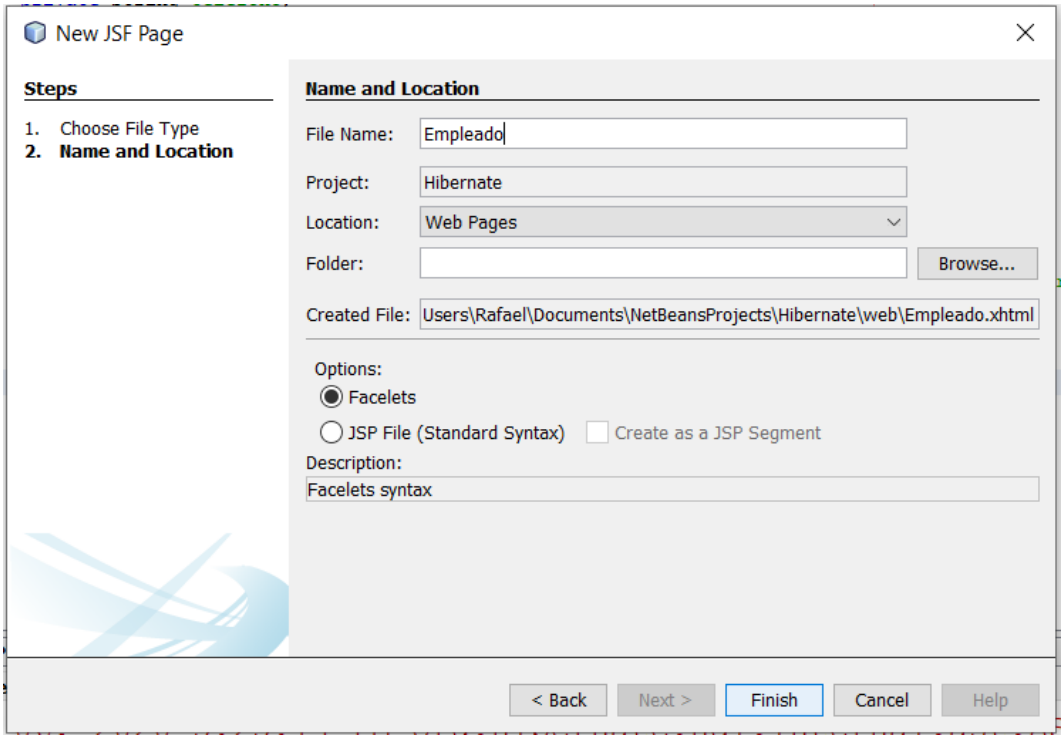
```
     */
    public String getDireccion() {
        return direccion;
    }

    /**
     * @param direccion the direccion to set
     */
    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

}
```

Paso 2.

- Crear una página JSF llamada Empleado, ver las siguientes imágenes



- Definir como nombre a la página JSF "Empleado", dejar seleccionada la opción de Facelets.



- Agregar el siguiente código a la página XHTML creada anteriormente.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets">
    <h:head>
        <title>Agregar Cliente</title>
    </h:head>
    <h:body>
      Agregar Cliente
       <h:form>
          <h:panelGrid columns="2">
             <h:outputLabel  for="id" value="ID: " /> <h:inputText id="id" va-
lue="#{empleadoBean.id}"/>
             <h:outputLabel  for="nombres" value="Nombres: " /> <h:inputText
id="nombres" value="#{empleadoBean.nombres}"/>
             <h:outputLabel  for="nombres" value="Apellidos: " /> <h:inputText
id="apellidos" value="#{empleadoBean.apellidos}"/>
             <h:outputLabel  for="edad" value="Edad: " /> <h:inputText id="edad" va-
lue="#{empleadoBean.edad}"/>
             <h:outputLabel  for="tel" value="Telefono: " /> <h:inputText id="tel" va-
lue="#{empleadoBean.telefono}"/>
```

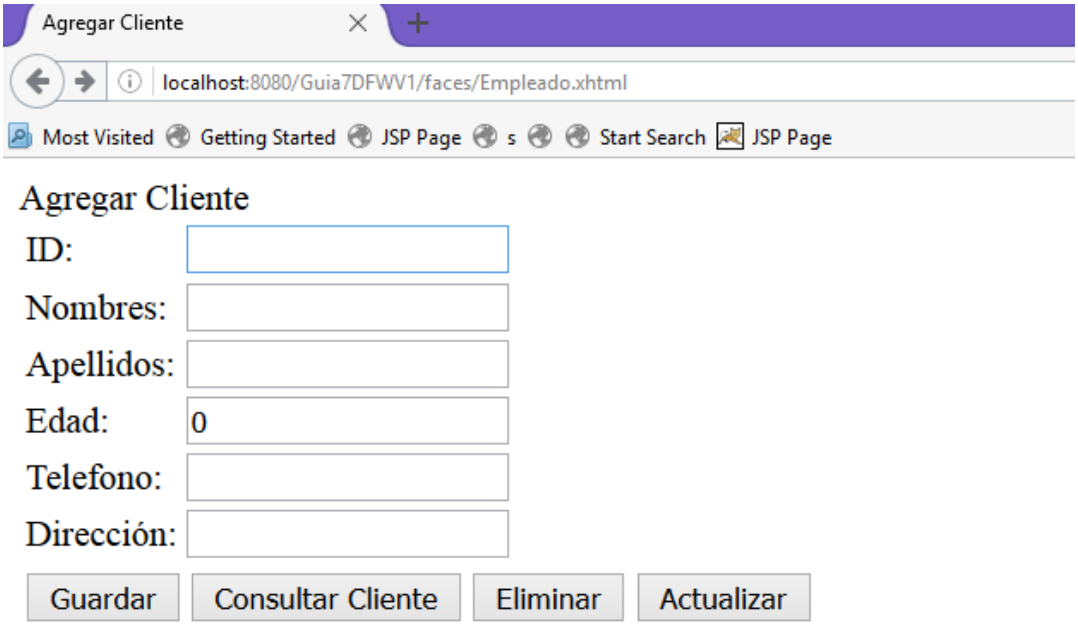
```
            <h:outputLabel  for="direc" value="Dirección: " /> <h:inputText id="direc"
value="#{empleadoBean.direccion}"/>

          </h:panelGrid>

          <h:panelGrid columns="4">
             <h:commandButton value="Guardar" ac-
tion="#{empleadoBean.addEmpleado()}" />
             <h:commandButton value="Consultar Cliente" ac-
tion="#{empleadoBean.returnEmpleadoId()}" />
             <h:commandButton value="Eliminar" ac-
tion="#{empleadoBean.deleteEmpleado()}" />
             <h:commandButton value="Actualizar" ac-
tion="#{empleadoBean.updateEmpleado()}" />
          </h:panelGrid>

          <h:messages infoStyle="color: blue" globalOnly="true"/>
       </h:form>
      <br/>
       <h:dataTable var="list" value="#{empleadoBean.empleados}" >
           <h:column>
          <f:facet name="header">
             <h:outputText value="ID"/>
          </f:facet>
           <h:outputText value="#{list.id}"/>
          </h:column>
          <h:column>
           <f:facet name="header">
             <h:outputText value="Nombres"/>
          </f:facet>
            <h:outputText value="#{list.nombres}"/>
           </h:column>
          <h:column>
           <f:facet name="header">
             <h:outputText value="Apellidos"/>
          </f:facet>
             <h:outputText value="#{list.apellidos}"/>
          </h:column>
          <h:column>
          <f:facet name="header">
             <h:outputText value="Edad"/>
          </f:facet>
             <h:outputText value="#{list.edad}"/>
          </h:column>
          <h:column>
           <f:facet name="header">
             <h:outputText value="Dirección"/>
          </f:facet>
            <h:outputText value="#{list.nombres}"/>
           </h:column>
       </h:dataTable>
```

```
   </h:body>
</html>
```

Resultado

Al ejecutar la aplicación se deberá ver un resultado, similar al siguiente el cual tendrá 4 opciones disponibles:

- **Guardar:** Permite Crear un Cliente Nuevo (No se debe agregar el ID).
- **Consultar Cliente:** Permite hacer la búsqueda un cliente en especifico (Agregar el ID del empleado y luego dar en Consultar, esto llenara todos los campos con la información).
- **Eliminar:** Solamente se necesita el ID del empleado para eliminarlo de la Base de Datos.
- **Actualizar:** Se necesita el ID del empleado, este realizar una búsqueda en la base de datos actualizara la información con la que se ingrese en el formulario.



ID	Nombres	Apellidos	Edad	Dirección
1	Carlos	Flores	19	Zacamil, Apopa
2	Kenia	Hernandez	22	Altavista, San Martin
5	Carlos	Quintanilla	19	Zacamil, Mejicanos
6	Karens Elena	Alfaro	23	Las Margaritas, Soyapango
8	Laura Carolina	Ramirez	23	Sierra Morena, Soyapango

IV. Ejercicios Complementarios

Crear una aplicación que tenga 2 tablas, las cuales serán:
-Producto
-Categoría
Deberá crear un CRUD para ambas tablas, tomar en cuenta que estas deberán estar relacionadas.

V. REFERENCIA BIBLIOGRAFICA

- http://www.javahispano.org/contenidos/es/manual_hibernate/
  Fecha de última visita: 26/09/2010.

- http://www.davidmarco.es/tutoriales/hibernate-reference/index.html#tutorial-firstapp
  Fecha de última visita: 26/09/2010.

- http://yaqui.mxl.uabc.mx/~larredondo/distribuidas/Hibernate.htm
  Fecha de última visita: 26/09/2010.

- INTRODUCCIÓN A HIBERNATE
  Autor: Francesc Rosés Albiol

- Manual Hibernate
  Autor: Héctor Suárez González

HOJA DE EVALUACIÓN

Hoja de cotejo: 6

Alumno:	Carnet:
Docente:	Fecha:
Título de la guía:	No.:

Actividad a evaluar	Criterio a evaluar	Cumplió		Puntaje
		SI	NO	
Discusión de resultados G8	Realizó los ejemplos de guía de práctica (40%)			
	Presentó todos los problemas resueltos (20%)			
	Funcionan todos correctamente y sin errores (30%)			
	Envió la carpeta comprimida y organizada adecuadamente en subcarpetas de acuerdo al tipo de recurso (10%)			
	PROMEDIO:			
Investigación complementaria G8	Envió la investigación complementaria en la fecha indicada (20%)			
	Resolvió todos los ejercicios planteados en la investigación (40%)			
	Funcionaron correctamente y sin ningún mensaje de error a nivel de consola o ejecución (4 0%)			
	PROMEDIO:			