IES CHAN DO MONTE
C.S. de Desarrollo de Aplicaciones Multiplataforma

ANEXO I:

UNIDAD 4: HIBERNATE –HERRAMIENTA ORM-

Índice

1. TA1. Ficheiro Properties

Os arquivos coa extensión `.Properties` son utilizados en java, xeralmente para almacenar parámetros configurables dunha aplicación como configuración de directorios, aspecto, valores de configuración, direccións web, etc. que o usuario configura con respecto á aplicación para personalizala ao seu gusto. Estes ficheiros de propiedades teñen formato de texto plano. Pódense crear con calquera editor de textos.

Na plataforma Java existe unha clase pensada para almacenar, recuperar e xestionar estas propiedades, é a `clase Properties`.

Cada parámetro dun arquivo.Properties, almacénase en dúas partes, a primeira a *KEY (clave)* e o segundo o *valor do parámetro*.

- Clave: sérvenos para identificar noso parámetro en Java
- Valor: é o valor almacenado nese parámetro.

Para os comentarios utilízase o símbolo " # ".

Exemplo dun arquivo de propiedades `Datos.properties`

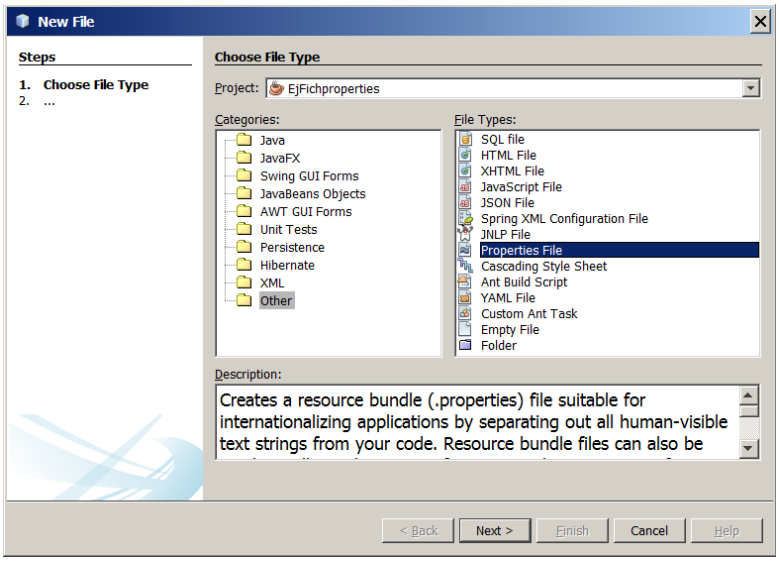
```
# Ficheiro de propiedades Datos.properties
#Parámetros do usuario
nome=Manuel
idioma=Español
```

A través do método `getProperty`, pódese acceder aos valores das propiedades solicitadas, e mediante o método `setProperty` pódense fixar valores para propiedades xa existentes, ou ben crear novas propiedades.

```
    public static void main(String[] args) throws IOException {
        Properties p = new Properties();
        //para cargar o ficheiro de propiedades
        p.load(new FileInputStream("src/ejfichproperties/Datos.properties"));
        //obten o valor da propiedade
        System.out.println("Nome de usuario: " + p.getProperty("nome"));
        System.out.println("Idioma de usuario: " + p.getProperty("idioma"));
        //para cambiar o valor dunha propiedade
        p.setProperty("nome", "Sergio");
        System.out.println("Nome de usuario: " + p.getProperty("nome"));
        //Almacena o novo valor do parámetro nome no ficheiro
        p.store(new FileOutputStream("src/ejfichproperties/Datos.properties"), "nome");
    }
```

```
ut - EjFichproperties (run)
run:
Nome de usuario: Manuel
Idioma de usuario: Español
Nome de usuario: Sergio
BUILD SUCCESSFUL (total time: 1 second)
```

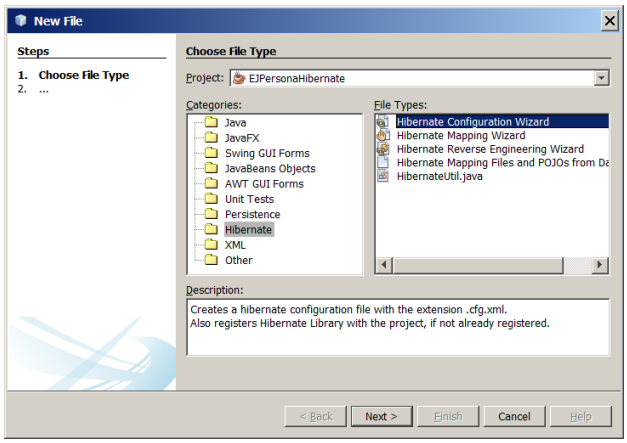
No IDE Netbeans pódese crear un novo `arquivo.Properties` no proxecto, para isto, facemos clic co botón dereito sobre o paquete onde se crease o arquivo, escollemos `New - > Other`. Na ventá que aparece, escollemos "`Properties File`" e dámoslle un nome único ao arquivo.



2. Utilización do IDE Netbeans para crear unha aplicación utilizando Hibernate

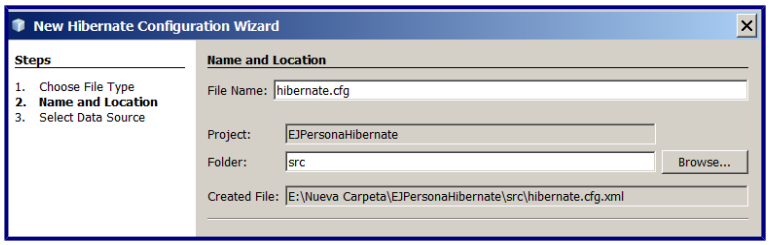
- Creamos un novo proxecto de NetBeans.
  - Menú "`File -> New Project... -> Java -> Java Application`". Dámoslle un nome e unha situación ao proxecto e asegurámonos de que as opcións "`Create Main Class`" e "`Set as Main Project`" estean habilitadas. Prememos o botón "Finish" e veremos aparecer no editor a nosa clase "`Main`".
- Agregamos a biblioteca de Hibernate, ao noso proxecto.
  - Facemos clic dereito no nodo "`Libraries`" do proxecto. No menú contextual que se abre seleccionamos a opción "Add Library... ":
  - Engadimos o conectador da base de datos que utilicemos para conectarnos no noso proxecto
- Para crear os arquivos necesarios de Hibernate noso proxecto, seleccionamos "new File/Hibernate".

Temos un asistente para crear cada arquivo necesario no proxecto.



2.1 Crear o arquivo de configuración (`xxxx.cfg.xml`)

`Hibernate Configuración Wizard` é un asistente para axudar a crear o arquivo de configuración de hibernate en XML. Por defecto, proponnos o nome hibernate.cfg.xml. O ficheiro de configuración debe estar no directorio raíz da nosa aplicación.



O seguinte paso sería indicar a conexión á base de datos que creamos no nodo "`services/database/new connection`".

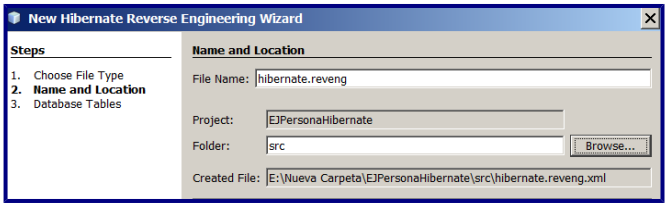
O ficheiro hibernate.cfg.xml será engadido ao noso proxecto.

2.2 Crear o arquivo de enxeñaría inversa (`hibernate.reveng.xml`)

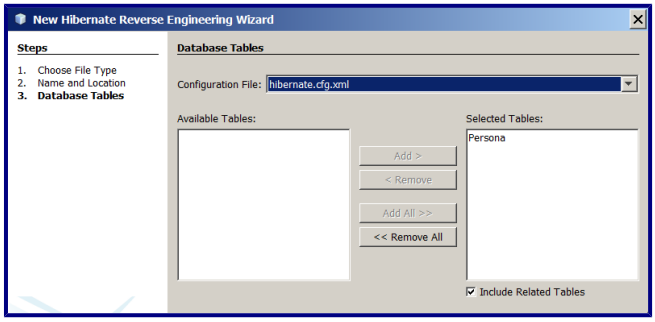
Si se desexa utilizar os arquivos de mapeo Hibernate e POJOS dun asistente de base de datos, primeiro débese crear un arquivo de hibernate.reveng.xml enxeñaría inversa.

Os arquivos de mapeo Hibernate e POJOS dun asistente de base de datos requiren hibernate.reveng.xml e Hibernate.cfg.xml.

Este arquivo ten que estar no mesmo directorio que o ficheiro de configuración hibernate.cfg.xml. Neste arquivo gardase a configuración das nosas táboas. Utilizaremos o asistente "`Hibernate Reverse Engineering Wizard`" para construír este ficheiro.



Especificamos as táboas que se van a mapear.

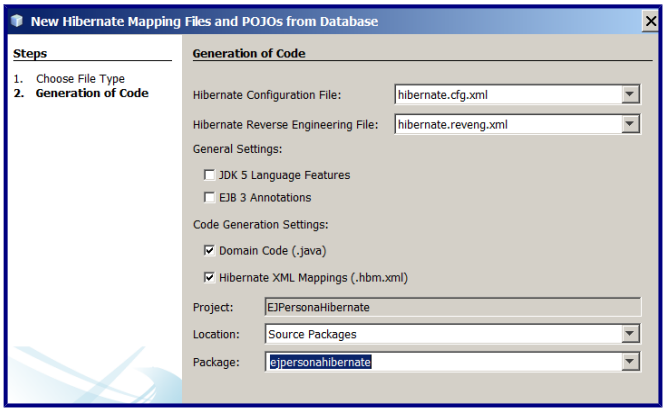


Créanos o ficheiro hibernate.reveng.xml co seguinte formato:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-reverse-engineering PUBLIC "-//Hibernate/Hibernate Reverse Engineering
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-reverse-engineering-3.0.dtd">
<hibernate-reverse-engineering>
  <schema-selection match-catalog="Persoas"
                    match-schema="dbo"/>
  <table-filter match-name="Persoa"/>
</hibernate-reverse-engineering>
```

2.3 Crear os arquivos de mapeo e POJOS.

Para crear os arquivos de mapeo e os POJOS utilizamos o asistente "`Hibernate Mapping Files and POJOs from Database wizard`".



Permite especificar as táboas nunha base de datos e xerar os arquivos de asignación e POJOS baseados nas táboas da base de datos. Se non desexamos utilizar os arquivos de asignación xml de Hibernate, o asistente danos a opción de crear clases que utilizan as anotacións para a información de asignación.

Debemos proporcionar o *Arquivo de configuración*, por exemplo, `hibernate.cfg.xml` (si é o utilizado por defecto) e o *ficheiro de enxeñaría inversa Hibernate* (`hibernate.reveng.xml`).

- En `General Settings`:

- Si se activa `JDK 5 Language Features`, actívanse as características da linguaxe Java compatibles co JDK 5 EJB 3.
- Si se activa `EJB 3 Annotations`, creanse os POJOS con anotacións para facer o mapeo.
  - En `Code Generation Settings`, a configuración de xeración de código determinará que arquivos se xeran:
- Cando se selecciona `Domain code (. Java)`, o IDE Netbeans *xera un POJO para cada unha das táboas* que se especificaron no asistente. Cando esta opción non está seleccionada, o asistente non xerará ningún POJO e teremos que crealos de forma manual.
- Cando se selecciona `Hibernate XML mappings (. hbm.xml)`, o IDE *xera un arquivo de asignación (hbm.xml)* para cada unha das táboas que se especificaron no asistente.

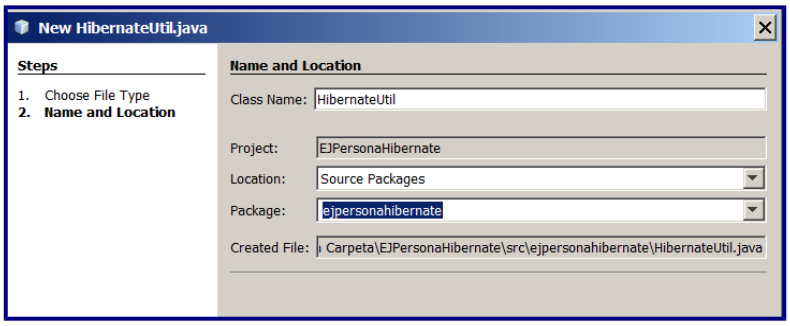
Na seguinte imaxe móstrase un exemplo dun POJO para almacenar información de persoas. A clase definida (POJO) debe seguir as anotacións Javabeans: Ten que ter un construtor por defecto e os seus atributos deben ser privados. A estos atributos accedense ou modifícanse a través dos seus métodos getter e setter.

```
public class Persoa implements Serializable {
    private int id;
    private String nome;
    private String apelido1;
    private String apelido2;
    private String email;
    private String telefono;
    Persoa() {    }
    Persoa(int id, String nome, String apelido1, String apelido2,
            String email, String telefono) {
        this.id = id;
        this.nome = nome;
        this.apelido1 = apelido1;
        this.apelido2 = apelido2;
        this.email = email;
        this.telefono = telefono;
    }

    public String getApelido1() {      return apelido1;    }
    public void setApelido1(String apelido1) {  this.apelido1 = apelido1;  }
    public String getApelido2() {    return apelido2;    }
    public void setApelido2(String apelido2) {    this.apelido2 = apelido2;    }
    public String getEmail() {        return email;    }
    public void setEmail(String email) {        this.email = email;    }
    public int getId() {        return id;    }
    public void setId(int id) {  this.id = id;    }
    public String getNome() {        return nome;    }
    public void setNome(String nome) {        this.nome = nome;    }
    public String getTelefono() {        return telefono;    }
    public void setTelefono(String telefono) {        this.telefono = telefono;    }
}
```

2.4 Crear a clase HibernateUtil.java

Imos xerar a clase que accede a `SessionFactory` para obter un obxecto `Session` para comunicarnos con Hibernate. Esta clase só se instancia unha vez e proporciona un punto de acceso global a ela. Para xerar esta clase, pódese utilizar o asistente `Hibernate Util`.



Xérase a clase Hibernate Util co o seguinte formato:

```
package Persoahibernate;
import org.hibernate.cfg.AnnotationConfiguration;
import org.hibernate.SessionFactory;
public class HibernateUtil {
    private static final SessionFactory sessionFactory;

    static {
        try {   sessionFactory =
                    new AnnotationConfiguration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Log the exception.
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }
public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

A partir de hibernate 3.6, `AnnotationConfiguration` *está deprecated*. Hai que cambialo e utilizar `Configuration` no seu lugar.

```
sessionFactory = new Configuration().configure().buildSessionFactory();
```

A partir de agora, cada vez que necesitemos acceder á base de datos, crearemos un obxecto `Session`. Exemplo:

```
Session sesion=HibernateUtil.getSessionFactory().openSession();
Persoa pers;
pers=(Persoa) sesion.get(Persoa.class,4);
System.out.println("Nome:"+pers.getNome());
```