Hibernate

Hibernate, como la definen sus autores, es una herramienta de mapeo objeto/relacional para ambientes Java.
No solo se encarga del mapeo de clases Java a tablas de la base de datos (y de regreso), sino que también maneja los queries y recuperación de datos, lo que puede reducir de forma significativa el tiempo de desarrollo que de otra forma gastaríamos manejando los datos de forma manual con **SQL**.

Entonces, si cambiamos el manejador de base de datos no será necesario que modifiquemos todo el SQL que ya teníamos, para adaptarse al SQL que maneja la nueva base de datos. Solo será necesario modificar una línea en un archivo de configuración de Hibernate, y este se encargará del resto.

Antes de seguir, veamos los siguientes temas que necesitamos conocer para manejar Hibernate:

¿QUÉ ES UN POJO, Y UN BEAN?

POJO son las iniciales de "Plain Old Java Object", que puede interpretarse como "Un objeto Java Plano Antiguo". Un POJO es una instancia de una clase que no extiende ni implementa nada en especial.
Para los programadores Java sirve para enfatizar el uso de clases simples y que no dependen de un framework en especial. Este concepto surge en oposición al modelo planteado por los estándares EJB anteriores al 3.0, en los que los Enterprise JavaBeans (EJB) debían implementar interfaces especiales.

Un POJO no debe contener lo siguiente:

- Extender clases preespecificadas, Ej: `public class GFG extends javax.servlet.http.HttpServlet {...}` no es una clase POJO.

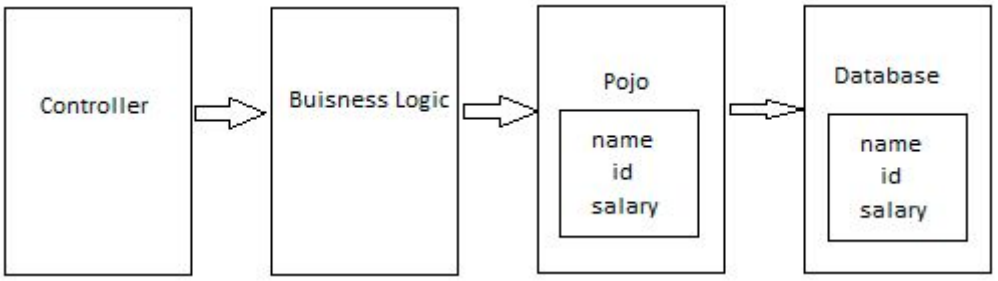
- Implementar interfaces preespecificadas, Ej: `public class Bar implements javax.ejb.EntityBean {...}` no es una clase POJO.
- Anotaciones preespecificadas, Ej: `@javax.persistence.Entity public class Baz {...}` no es una clase POJO.

POJOs básicamente define una entidad. Si en tu programa deseas una clase de empleado, entonces puedes crear un POJO de la siguiente manera:

```
// Employee POJO class to represent entity Employee
public class Employee
{
    // default field
    String name;

    // public field
    public String id;

    // private salary
    private double salary;

    //arg-constructor to initialize fields
    public Employee(String name, String id,
                            double salary)
    {
        this.name = name;
        this.id = id;
        this.salary = salary;
    }

    // getter method for name
    public String getName()
    {
        return name;
    }

    // getter method for id
    public String getId()
    {
        return id;
    }

    // getter method for salary
    public Double getSalary()
    {
        return salary;
    }
}
```

El ejemplo anterior es un ejemplo bien definido de la clase POJO. Como puede ver, no hay restricción sobre el acceso-modificador de los campos. Pueden ser privados, predeterminados, protegidos o públicos. Tampoco es necesario incluir ningún constructor en él.

POJO es un objeto que encapsula la lógica de negocio (Business Logic). La siguiente imagen muestra un ejemplo práctico de la clase POJO. Los controladores interactúan con su lógica de negocio, que a su vez interactúan con POJO para acceder a la base de datos. En este ejemplo, una entidad de base de datos está representada por POJO. Este POJO tiene los mismos miembros que la entidad de la base de datos.



Un JavaBean (o también conocido simplemente como Bean) es una clase simple en Java que cumple con ciertas normas. Los JavaBean, son un modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones en Java.
Se usan para encapsular varios objetos en un único objeto (la vaina o Bean en inglés), para hacer uso de un solo objeto en lugar de varios más simples.
La especificación de JavaBeans de Sun Microsystems los define como "componentes de software reutilizables que se puedan manipular visualmente en una herramienta de construcción".

Convenciones JavaBean

Para funcionar como una clase JavaBean, una clase debe obedecer ciertas convenciones sobre nomenclatura de métodos, construcción y comportamiento.
Estas convenciones permiten tener herramientas que puedan utilizar, reutilizar, sustituir y conectar JavaBeans.
Las convenciones requeridas son:

- Debe tener un constructor sin argumentos.

- Sus atributos de clase deben ser privados.
- Sus propiedades deben ser accesibles mediante métodos get y set que siguen una convención de nomenclatura estándar.
- Debe ser serializable.

Estructura

Dentro de un JavaBean podemos distinguir tres partes:

- Propiedades: Los atributos que contiene.
- Métodos: Se establecen los métodos get y set para acceder y modificar los atributos.
- Eventos: Permiten comunicar con otros JavaBeans.

Ejemplo

```
public class PersonaBean
  implements java.io.Serializable {

  private String nombre;
  private int edad;

  public PersonaBean() {
    // Constructor sin argumentos
  }

  // Constructor del JavaBean   public PersonaBean(String nombre, int edad)
{
    this.nombre = nombre;
    this.edad = edad;
  }

  // Constructor por copia
  public PersonaBean(PersonaBean personaBean) {
    this.nombre = personaBean.getNombre();
    this.edad = personaBean.getEdad();
  }

  public void setNombre(String n) {
   this.nombre = n;
  }
```

```
  public void setEdad(int e) {
    this.edad = e;
  }

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

  public int getEdad() {
   return (this.edad);
  }
}
```

POJO hace referencia a Plain Old Java Object. Esto se refiere a una clase con atributos privados o protegidos y que solo provee métodos getter y setter protegidos o públicos para acceder a los atributos. Similar a cuando defines una entidad.

Entonces, todos los JavaBeans son POJO pero no todos los POJOs son JavaBeans.

¿Qué es Serializable?

Serializable es una clase ubicada en el paquete **Java.io.Serializable**, la cual no cuenta con ningún método, por lo que es una clase que sirve solamente para especificar que todo el estado de un objeto instanciado podrá ser escrito o enviado en la red como una trama de bytes.

Para que un programa java pueda convertir un objeto en un montón de bytes y pueda luego recuperarlo, el objeto necesita ser Serializable. Al poder convertir el objeto a bytes, ese objeto se puede enviar a través de red, guardarlo en un fichero, y después reconstruirlo al otra lado de la red, leerlo del fichero,....

Si dentro de la clase hay atributos que son otras clases, éstos a su vez también deben ser **Serializable**. Con los tipos de *java* (**String**, **Integer**, etc.) no hay problema porque

lo son. Si ponemos como atributos nuestras propias clases, éstas a su vez deben implementar **Serializable**.

Los archivos Jar (Java ARchives) permiten recopilar en un sólo archivo varios archivos diferentes, almacenándolos en un formato comprimido para que ocupen menos espacio.

Es por tanto, algo similar a un archivo .zip (de hecho están basados en archivos .zip).

Entonces, ¿dónde está la "gracia"? ¿No se podrían usar directamente archivos .zip? La particularidad de los archivos .jar es que no necesitan ser descomprimidos para ser usados, es decir que el intérprete de Java es capaz de ejecutar los archivos comprimidos en un archivo jar directamente.

Por ejemplo, si hemos recopilado todos los ficheros necesarios para ejecutar una aplicación en un fichero "aplic.jar", podemos lanzar la aplicación desde una terminal de texto mediante:

```
java -jar aplic.jar
```

Preguntas??

Volvamos a Hibernate: existen dos formas de hacer los mapeos en Hibernate, la primera es a través de archivos de mapeo en XML

¿QUÉ ES EL LENGUAJE XML?

XML (Extensible Markup Language) es un lenguaje de etiquetas, es decir, cada paquete de información está delimitado por dos etiquetas como se hace también en el lenguaje HTML, pero XML separa el contenido de la presentación. Explicaremos esto con el siguiente ejemplo:

<H1>Mateo</H1> <--- HTML

<Nombre>Mateo</Nombre> <--- XML

<H1> y <Nombre> son etiquetas. Ambas encierran el texto o paquete de información “Mateo”. La etiqueta <H1> es de HTML, y se encarga de mostrar visualmente el texto “Mateo” en la página web en un tamaño determinado pero no dice nada del significado de “Mateo”: si es una ciudad o un nombre, por ejemplo. En cambio la etiqueta <Nombre> es de **XML** y nos dice que “Mateo” es un nombre de persona, por lo tanto **XML** se preocupa del significado del texto que encierra y no de la apariencia de cómo se muestre el texto en la página web.

Por eso se dice que XML es un lenguaje de etiquetas, que como hemos dicho anteriormente, separa el contenido de la presentación. Lo mismo se puede definir el lenguaje XML usando palabras más técnicas pero con el mismo significado que la definición anterior: **“XML describe el sentido semántico de los datos dejando de lado la presentación”.**

¿POR QUÉ ES ÚTIL EL LENGUAJE XML PARA LOS PROGRAMAS INFORMÁTICOS?

Un programa informático puede estar escrito en Java, Visual Basic y cualquier otro lenguaje. En esencia, todos los programas procesan información, entendiéndose por información “dato + significado”. Para el caso que estamos viendo, el dato en el ejemplo sería “Mateo” y el significado es un “nombre de persona”. Por lo tanto un documento escrito en XML tendría la información que necesitan los programas para procesar.

XML se plantea como un lenguaje estándar para el intercambio de información entre diferentes programas de una manera segura, fiable y libre, ya que no pertenece a ninguna compañía. Podemos ver por qué el XML es tan interesante para el intercambio de datos con el siguiente ejemplo:

Mateo nació el 15.10.2012 en la ciudad de Madrid con un peso de 3.1 kg y una estatura de 45 cm.

Maribel nació el 11.09.1976 en la ciudad de Sevilla con un peso de 3 Kg y una estatura de 40 cm.

Analizando el texto, nos encontramos que hay datos como “Madrid” y su correspondiente significado, que es una “ciudad” y otros más en un formato humano, tan sólo entendible por personas, no por los programas. Por tanto, podemos convertir el texto tanto en una base de datos “tradicional” como en un archivo o documento XML, que son formatos que los programas ya podrían entender, de la siguiente manera:

En formato tabla (base de datos “tradicional”):

Nombre	Fecha	Ciudad	Peso	Estatura
Mateo	15.10.2012	Madrid	3.1	45
Maribel	11.09.1976	Sevilla	3	40

En formato XML:

```
<Datos-Nacimiento>
  <Persona>
    <Nombre>Mateo</Nombre>
    <Fecha>15.10.2012</Fecha>
    <Ciudad>Madrid</Ciudad>
    <Peso>3.1Kg</Peso>
    <Estatura>45cm</Estatura>
  </Persona>
  <Persona>
    <Nombre>Maribel</Nombre>
    <Fecha>11.09.2012</Fecha>
    <Ciudad>Sevilla</Ciudad>
    <Peso>3Kg</Peso>
    <Estatura>40cm</Estatura>
  </Persona>
</Datos-Nacimiento>
```

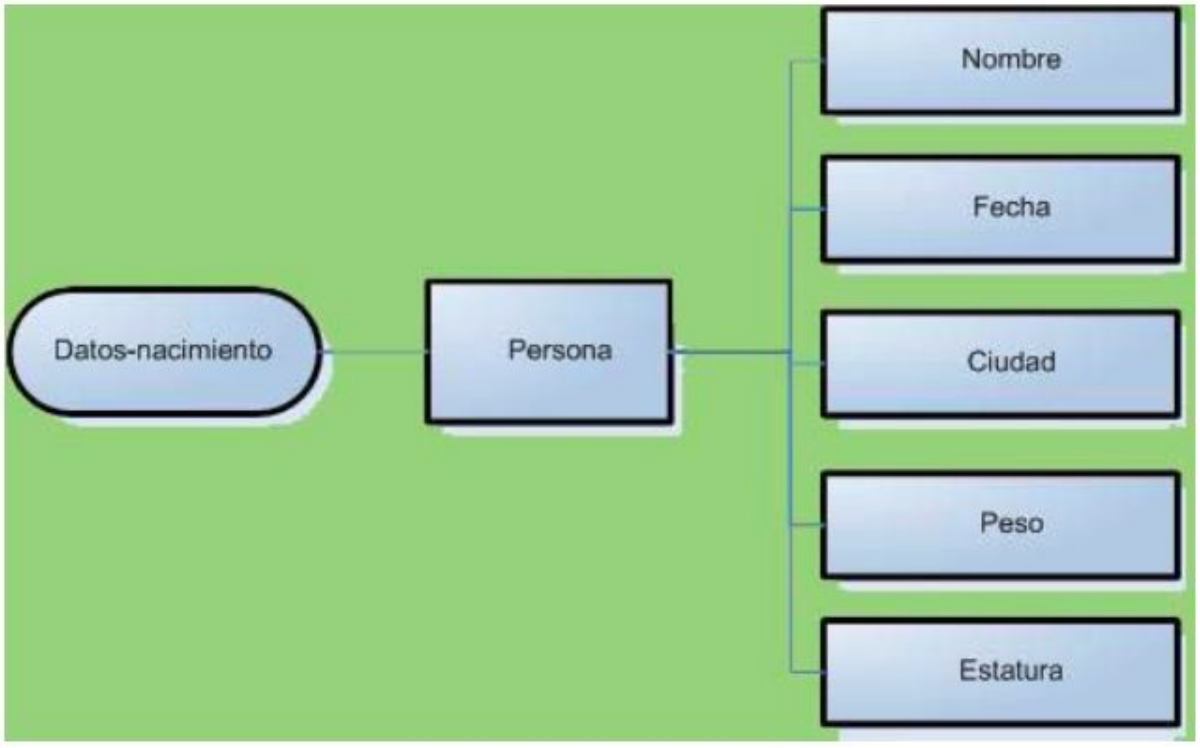
REGLAS BÁSICAS PARA ESCRIBIR CORRECTAMENTE UN DOCUMENTO XML

- Una etiqueta de apertura siempre tiene su contraparte de cierre con "/". Por ejemplo:

Incorrecto : <Nombre>Mateo

Correcto : <Nombre>Mateo</Nombre>

- Sólo puede haber un elemento raíz, en el que estén contenidos todos los demás, con una estructura jerárquica.



- El acrónimo "XML"(o "xml" o "xMl", etc.) no puede usarse como caracteres iniciales de un nombre de etiqueta o atributo.

- El XML es sensible al tipo de letra utilizado ("case-sensitive"), es decir, trata las mayúsculas y minúsculas como caracteres diferentes. Por ejemplo, no es lo mismo <automóvil> que <Automóvil>

- Una etiqueta vacía, es la que no tiene contenido, por lo que se cerraría al final en la misma etiqueta de apertura. Por ejemplo:

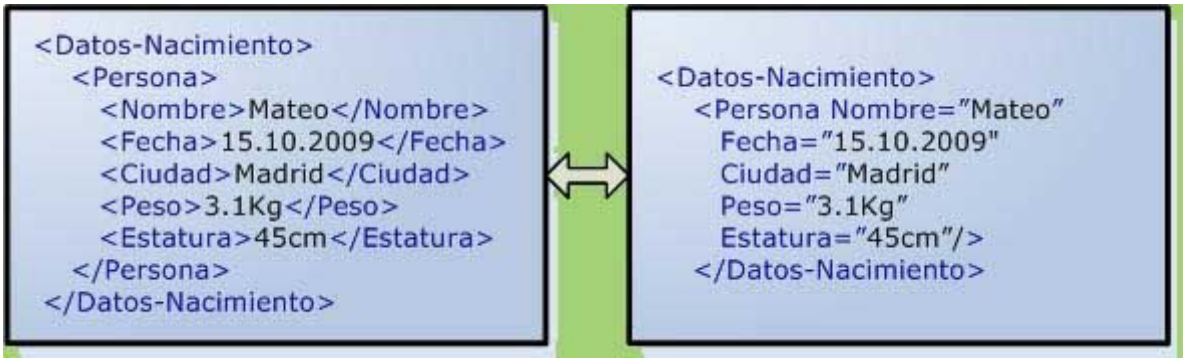
<Persona nombre=”walter”/>

<parámetro />

- Las etiquetas pueden tener atributos, que son una manera de incorporar características o propiedades a las etiquetas de un documento. El atributo consta de dos partes: La propiedad del elemento y el valor de la propiedad, que siempre va entre comillas doble (“) o simple (‘). Por ejemplo: modelo y color serian atributos de la etiqueta Vehiculo:

<Vehiculo marca=’Toyota’ modelo="45 TC" color="plomo">En venta</Vehiculo>

- Una etiqueta con contenido, puede modelarse como una etiqueta vacía con atributos. Por ejemplo:



Nota:

- Los archivos XML están codificados en texto plano, así que puedes abrirlos con cualquier editor de texto y podrás leerlos con toda claridad
- Es posible que veas el código <?xml version="1.0" encoding="UTF-8"?> en la parte superior. Esto significa que el contenido que aparece a continuación está en formato XML.
- Se pueden abrir con cualquier navegador, si lo abres con un navegador web será más fácil explorarlo. Esto se debe a que la mayoría de los navegadores aplican sangrías en las etiquetas automáticamente y te permiten contraer o expandir cada sección del árbol XML.

- Un archivo XML tiene la extensión xml: por ejemplo prueba.xml

Preguntas??