Algunas pregunta interesantes que deberías saber responder (Alberto)

**Implementar modelo en Java**

1. Define Entidad y ValueType. ¿Qué es mutable e inmutable en cada uno de ellos? ¿Sobre qué atributos se debe deben definir los métodos hashCode() y equals() en cada uno de ellos?

**Entidad**: Una entidad representa un concepto del dominio. Puede estar asociada con otras entidades. Su ciclo de vida es independiente. Debe tener una clave primaria (INMUTABLE). ¿El resto de atributos es lo que puede cambiar? (MUTABLE). Debe implementar Serializable.

Es sobre la clave natural en donde se define el hashCode() y el equals().

**Value type**: Representan información adicional, no conceptos principales de dominio. Se suelen presentar como atributos de una entidad o como composiciones (UML). Su ciclo de vida depende enteramente de la entidad que las posee.No pueden tener referencias entrantes.

1. ¿Por qué escribimos métodos para el mantenimiento de las asociaciones addXxxx(...) y removeXxxx(...) en vez de hacer las asignaciones directamente?

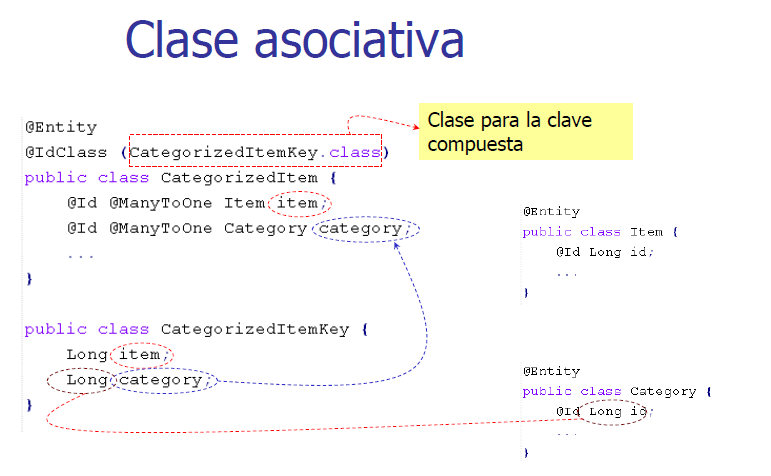
Porque puede que sea necesario hacer/deshacer esas asociaciones en otra parte del código.

1. ¿Cómo se implementa en Java una clase asociativa UML?

Association.Facturar.link(a, f);

Association.Facturar.unlink(a, f);

En java es una clase más, mapeada con dos relaciones muchos a uno y clave compuesta



1. ¿Qué parámetros mínimos debe recibir el constructor de una Entidad?

Tiene que tener un constructor sin parámetros y en otro constructor deberá recibir los parámetros que hagan que la entidad esté en un estado válido.

1. ¿Qué parámetros debe recibir el constructor de un ValueType?

Tiene que recibir tantos parámetros como atributos tenga.

1. ¿Qué métodos no hay que implementar en un ValueType?

Los Setter porque debe ser inmutable.

1. ¿Por qué los ValueTypes deben ser inmutables?

Para no generar inconsistencias, es decir, en Java generalmente las variables se pasan por referencia. Tratando los ValueTypes como elementos inmutables se evitan problemas al recibir en setters y valores erróneos al usar getters.

1. ¿Qué atributos no deben cambiar nunca en una Entidad?

Los que permiten distinguir a esa entidad de las demás, es decir, los que componen su clave primaria.

1. En la implementación de una entidad ¿A qué llamábamos atributos naturales? ¿Para qué es útil redefinir el método toString() de las clases del modelo?

A los que tienen significado en el contexto de uso (para el usuario: las entiende y las maneja) p.e. DNI, Nº de la SS, etc.

Es útil ya que al redefinir este método se visualiza de una manera más clara el estado en que está el objeto en un momento dado. Para Depurar

**Funcionamiento del mapeador**

1. Describe las diferencias entre los estados Transient, Persistent y Detached.

Transient: Un objeto recién creado que no ha sido enlazado con el gestor de persistencia (sólo existe en memoria de la JVM)

Persistent: Un objeto enlazado con la sesión. Todos los cambios que se le hagan serán persistentes.

Detached: Un objeto persistente que sigue en memoria después de que termina la sesión: existe en java y en la BDD.

Un mapeador de objetos a relacional (O/R) debe resolver las diferencias entre los dos paradigmas.

1. ¿Cómo resuelve un mapeador JPA la herencia?

De tres formas:

* Tabla única para toda la jerarquía: InheritanceType.SINGLE\_TABLE.

Todas las clases persisten en una única tabla con la unión de todas las columnas de todas las clases.

* Tabla por cada clase no abstracta: InheritanceType.TABLE\_PER\_CLASS.

Una tabla por cada clase no abstracta.

* Tabla por cada clase: InheritanceType.JOINED.

Cada clase su propia tabla, la relación de herencia se resuelve con FKs. Cada tabla solo tiene columnas para las propiedades no heredadas.

Ventajas

* Es simple y eficiente
* Soporta el polimorfismo
* Fácil de implementar
* Fácil modificar cualquier clase

Desventajas

* Todas las columnas no comunes deben ser nullables
* Pueden quedar columnas vacías

1. ¿Cómo resuelve la diferencia de granularidad?

Distinguiendo entre Entidades (tienen identidad propia y participan en asociaciones) y Value Types que no necesitan tener identidad, sólo su valor (no participan en asociaciones y su ciclo de vida está ligado al de la clase).

1. ¿Cómo resuelve la detección de la identidad?

La identidad de un objeto viene dada por los atributos que formen parte del @id.

1. ¿Cómo resuelve la cardinalidad MANY to MANY de las asociaciones?

Es una asociación bidireccional donde se tienen vincular los dos extremos de la asociación mediante @ManyToMany. Sólo hace falta poner el mappedBy=”elemento” en uno de los extremos.

1. ¿Cómo resuelve la navegación?

Se busca crear la ilusión de que todos los objetos ya están en memoria.

Eager loading (precargar): Se carga un objeto y sus asociados. Puede cargar más objetos de la cuenta. Riesgo de producto cartesiano.

Lazy loading (bajo demanda): Se carga al necesitarlo. Puede generar demasiadas SELECT \* FROM. El problema de las n+1 consultas.

1. ¿Cómo resuelve la concurrencia?

* Protocolos pesimistas (prevención)
* Técnicas de bloqueo

Bloqueo binario: Muy restrictivo. Una única transacción puede poseer un bloqueo en un momento dado. Bloqueo de modo múltiple

Bloqueos compartidos/exclusivos (o de lectura/escritura). Un elemento bloqueado para lectura tiene un bloqueo compartido por varias transacciones. Un elemento bloqueado para escritura tiene un bloqueo exclusivo (sólo una transacción).

* Marcas de tiempo
* Multiversión

* Protocolos optimistas (corrección)
* Validación o certificación

1. En un programa concurrente ¿cuantas copias parciales del grafo habrá en memoria?

1. ¿Qué estrategias usan los mapeadores O/R para recrear en memoria una zona del grafo?

Respuesta de la pregunta 15.

1. Para acelerar el acceso al grafo los mapeadores implementan una caché. ¿Qué características tiene esa caché?

Optimiza el rendimiento al reducir el trasiego con la BBDD

Permite hacer optimizaciones: Write-behind delayed, Batch load/update

¿Caché por proceso?, ¿por hilo?, ¿por cluster?

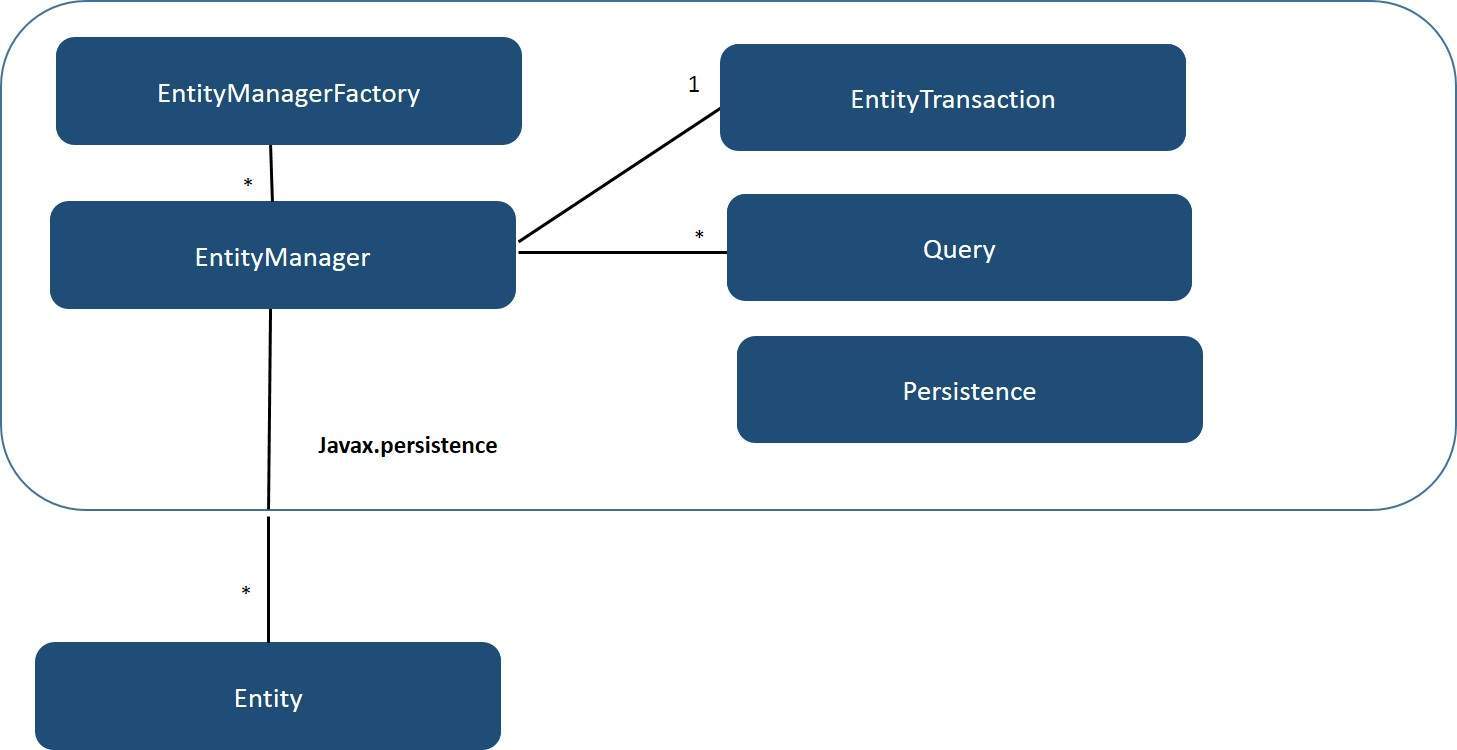
1. En este orden: se crea una instancia del mapeador, se abre transacción, se recupera un objeto con el método find() y se invoca a unos cuántos métodos del objeto recuperado que le producen modificaciones. ¿Qué hay que hacer después para actualizar la base de datos con los cambios?

Nada porque al recuperar un objeto con el método find() su estado es Persistent y todo cambio en él se ve reflejado en la base.

1. Describe brevemente las clases (o interfaces) principales que ofrece la API JPA: cuáles son, qué misión tienen, y cómo se interrelacionan.

|  |  |
| --- | --- |
| **EntityManagerFactory** | Esta es una clase de fábrica de EntityManager. Crea y gestiona múltiples instancias EntityManager. |
| **EntityManager.** | Es una interfaz, que gestiona la persistencia de objetos. Funciona como una factoría para instancias de la clase Query. |
| **Entity** | Las entidades son los objetos de persistencia, tiendas como registros en la base de datos. |
| **EntityTransaction** | Tiene una relación de uno a uno con EntityManager. Para cada EntityManager, se mantienen operaciones por la clase EntityTransaction. |
| **Persistence** | Esta clase contiene métodos estáticos para obtener instancias de EntityManagerFactory. |
| **Query** | Esta interfaz es implementada por cada proveedor JPA para obtener objetos relacionales que cumplan los criterios. |

Cómo se interrelacionan.



* La relación entre EntityManagerFactory EntityManager es de **uno a varios**. Se trata de una clase de fábrica para instancias de EntityManager.
* La relación entre método EntityManager y EntityTransaction es **uno a uno**. Para cada operación de EntityManager, hay una instancia de EntityTransaction.
* La relación entre EntityManager y Query es de **uno a varios**. Muchas consultas se pueden ejecutar mediante una instancia EntityManager.
* La relación entre Entity y EntityManager es **uno de muchos**. Una instancia de EntityManager puede administrar varias Entities.

1. ¿Se puede navegar por el grafo una vez cerrado el contexto de persistencia? Matiza la respuesta.

No. Se supone que dentro del contexto de persistencia está cargado todo el grafo pero realmente no es así, se va cargando según lo vamos recorriendo. Cuando el contexto se cierra, lo que se haya cargado hasta entonces permanece en memoria, pero NO se puede cargar más grafo. Por lo tanto si, en esas circunstancias, se intenta recorrer más grafo del que ya está cargado en memoria te salta LazyInitializationException. Sólo se podrá navegar por las partes del grafo que han sido cargadas.

1. ¿A que puede ser debido que me salte una LazyInitializationException?

A que esté navegando por una zona del grafo que no se ha cargado.

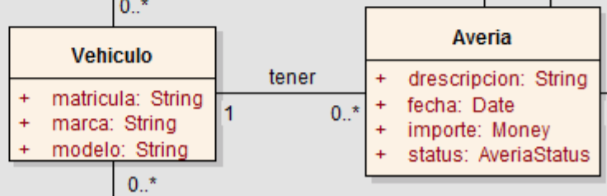
**Mapeo**

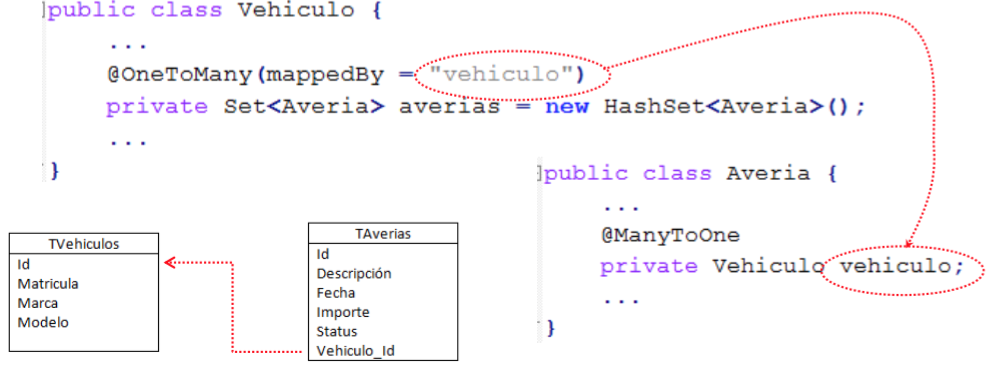
1. ¿De qué formas se puede añadir información de mapeo en JPA?

Mediante anotaciones. Empiezan por @ y pueden indicar mucha información útil para el mapeo (Indicar que: una clase es una entidad, vincular dos extremos de una asociación, un atributo forma parte de la clave primaria, establecer el tipo de herencia que se quiere usar, etc).

1. Añadiendo anotaciones de mapeo, ¿cómo se vinculan los dos extremos de una asociación bidireccional?, ¿qué pasa si no se hace?

En una asociación bidireccional se tienen vincular los dos extremos de la asociación. Depende de si es uno a muchos o muchos a muchos. Para el primer caso, en la parte del 1 tiene que haber una colección de elementos del tipo donde esté la n y en la parte n tiene que haber un objeto del tipo donde esté el 1. El mappedBy=”elemento” se pone en la colección indicando el nombre de la variable por la que se va a mapear.





Si no se indica mappedBy se interpreta como dos asociaciones unidireccionales separadas.

1. En JPA, con configuración de mapeo por defecto ¿los extremos MANY de una asociación se cargan de forma agresiva o bajo demanda?, ¿y los extremos ONE?

Por defecto se cargan bajo demanda todas las relaciones excepto OneToOne y ManyToOne.

Los extremos many se cargan bajo demanda (Lazy loading) y los extremos one se cargan de forma agresiva (Eager loading).

1. Un alumno se matrícula en varias asignaturas. En una asignatura se matricularán muchos alumnos. Para cada asignatura matriculada se obtiene una nota final. Dibuja el modelo UML. Escribe el código java mínimo para representar esa situación y las anotaciones de mapeo necesarias.
2. JPA, Hibernate, Eclipse Link. Explica qué relación hay entre esos términos.

JPA es la API de persistencia de Java que permite mantener las ventajas de la orientación a objetos al interactuar con una base de datos. Hibernate (GNU LGPL) y Eclipse Link (eclipse) son dos implementaciones para esta API.

1. Tenemos una aplicación desarrollada con mapeador JPA Hibernate funcionando contra una base de datos MySQL. Por alguna razón hay que cambiar de servidor de base de datos, ahora usaremos una Oracle. ¿Cómo lo resolvemos?

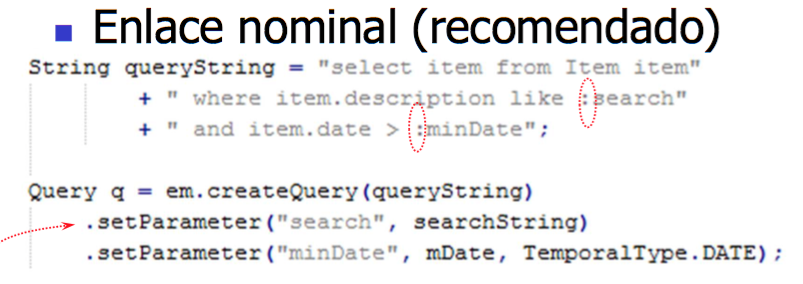
Se cambia el atributo jta-data-source en el archivo persistence.xml al nuevo servidor.

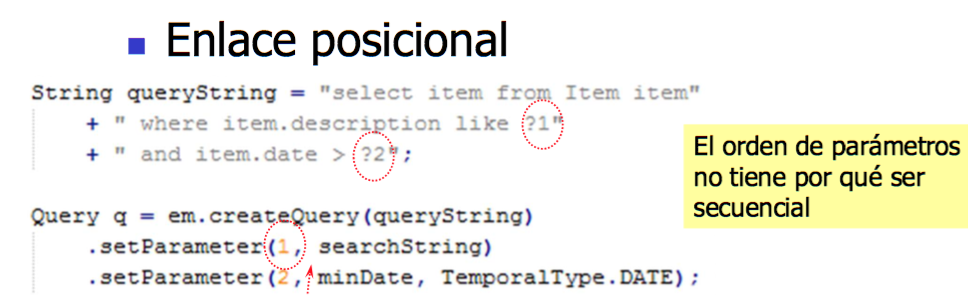
Then, in the persistence.xml, you need to change the jta-data-source attribute to point to the new data source in JNDI (e.g., java:/MysqlDS if you are using the default mysql-ds.xml to setup a MySQL external database).

**Consultas JPQL**

1. ¿De qué formas se pueden especificar y asignar los parámetros de una consulta?

De dos formas:





1. ¿Cómo se llama ese estilo de configuración de una Query a base de llamar a métodos que siempre devuelven el mismo objeto?

JPA posee las interfaces Query y TypedQuery para configurar y ejecutar consultas. La interfaz Query se utiliza para los casos en los que los resultados se esperan que sean de tipo Objeto. La interfaz TypeQuery extiende a Query, por lo que una consulta con tipo puede ser tratada como si fuera una consulta sin tipo, pero no al revés. Se obtiene una implementación de la interfaz apropiada para una consulta dada a través de uno de los métodos factoría de la interfaz del EntityManager. La elección de un método depende del tipo de la consulta.

1. ¿En qué estado están los objetos devueltos por una consulta JQL?, ¿y el que devuelve el método find()?

If an entity class is used as a result class, the result entity objects are created in the [NEW state](http://www.objectdb.com/java/jpa/persistence/managed#Entity_Object_Life_Cycle), which means that they are not managed. Such entity objects are missing the JPA functionality of managed entity objects (e.g. transparent navigation and transparent update detection), but they are more lightweight, they are built faster and they consume less memory.

Cuando se obtiene un objeto con el método find() su estado es Persistent.

1. ¿Qué fallos encuentras en estas consultas en JQL sobre CarWorkshop?

select \* from TMecanicos m X

select m from Mecanicos m   
 select m from mecanicos m X  
 select \* from TMecanicos X

1. Sobre CarWorkshop ¿es correcta esta consulta? En caso negativo explica por qué no.

Select a.vehiculo.cliente  
 from Averia a  
 where a.factura.fecha = ‘12/12/2014’

Si es correcta .

1. Sobre CarWorkshop ¿es correcta esta consulta? En caso negativo explica por qué no.

select f.averias.vehiculo.cliente  
 from Factura f  
 where f.fecha = ‘12/12/2014’

No es correcta porque el final del camino no puede ser multivaluado, es decir, no se puede pasar a través de asociaciones oneToMany.

**Arquitectura**

1. Enumera los patrones que aparecen en la realización del CarWorkshop con mapeador.

**Modelo-vista-controlador (MVC):** Separa los [datos](https://es.wikipedia.org/wiki/Datos) y la [lógica de negocio](https://es.wikipedia.org/wiki/L%C3%B3gica_de_negocio) de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.

**Data Access Object (DAO):** Este patrón de diseño divide más las responsabilidades en la aplicación de tal forma que tendremos unas clases que se encargaran de la lógica de negocio y otras clases la responsabilidad de persistencia.

**Transaction Script**: Utiliza procedimientos de tal forma que cada procedimiento se encarga de gestionar o procesar una petición única desde la capa de presentación.

**Facade**: Se aplicará cuando se necesite proporcionar una interfaz simple para un subsistema complejo, o cuando se quiera estructurar varios subsistemas en capas, ya que las fachadas serían el punto de entrada a cada nivel.

**Command**: Permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor real de la misma. Para ello se encapsula la petición como un objeto, con lo que además facilita la parametrización de los métodos.

**Factory Method**: Consiste en utilizar una clase constructora (al estilo del Abstract Factory) abstracta con unos cuantos métodos definidos y otro(s) abstracto(s): el dedicado a la construcción de objetos de un subtipo de un tipo determinado.

1. Estoy escribiendo un comando en el que me están saliendo muchas líneas de código, ¿qué es muy probable que esté haciendo mal?

Poniendo instrucciones en una capa a la que no corresponden.

1. Imagina que el CarWorkshop que se ha desarrollado pide login y password a cada usuario que lo usa. Una vez verificada la identidad del usuario su login queda accesible en una variable global de hilo (un ThreadLocal). Ahora nos piden que modifiquemos el programa de forma que guarde en un fichero de log una línea <usuario, fecha, hora, método de la capa de servicio invocado> cada vez que un usuario realiza una acción (que guarde una auditoria). ¿En qué clase añadirías esa funcionalidad de forma que sólo se escriba una vez?

En las clases de servicio?

1. Suponiendo configuración de mapeo por defecto ¿funcionará el código mostrado a continuación? Explica los motivos.

Capa de presentación

AdminService as = ServicesFactory.adminServices();

Cliente c = as.findClientById( 123 );

for(Vehiculo v : c.getVehiculos()) {

Printer.print( v );  
 }

…

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Capa de servicio

class AdminServicesImpl {  
 ...

Cliente findClienteById(Long id) {

EntityManager em = getEntityManagerAndOpenTrx();

try {

return em.find(Cliente.class, id);

} finally {

closeTrxAndEntityManager();

}

}

}

Falla porque no se declara en ningún momento la variable “c” y además al acceder a la variable vehículos saltaría una excepción ya que no se encuentra cargado en memoria.

1. En el código de un comando: Tengo un objeto “Vehiculo” obtenido con un find() y necesito todas sus averías ¿cómo me hago con ellas? Comenta las posibilidades y señala la más sencilla.

Llamando al atributo averías con su get correspondiente. (o haciendo una consulta?)

1. En el código de una clase Action: Tengo un objeto “Vehiculo” obtenido con una llamada a la capa de lógica y necesito todas sus averías ¿cómo me hago con ellas? Comenta las posibilidades y señala la más sencilla.

Se podría tener en cuenta que se va a acceder a este atributo y cargar esa colección al obtener el objeto de la base.