Integración de datos

Traballo tutelado ORM

Parte 2: Implementación

1 Descrición xeral

O obxectivo principal desta fase consiste en programar a capa de negocio e a capa de persistencia da aplicación, e un pequeno conxunto básico de casos de proba JUnit.

A implementación debe incluír o mapeo das clases para o paso a relacional, usando anotacións JPA; e implementar un conxunto de clases *DAO* que doten de persistencia ao modelo, facendo uso da API estándar JPA.

AVISO: **non está permitido** usar a API nativa de Hibernate. **Debedes** usar a API estándar JPA.

Como axuda para a implementación, este enunciado vai acompañado de material adicional:

- Unha pequena aplicación, similar á que debes desenvolver, e completamente implementada, que xestiona *entradas de log* correspondentes a certos *usuarios*.
- Un documento que describe os aspectos fundamentais da aplicación.

Utilizade o código como punto de partida, e implementade as vosas propias clases do modelo, DAOs e casos de proba, substituíndo aos orixinais.

2 Implementación da capa de negocio

Codificade as clases de negocio de acordo coas seguintes condicións **OBRIGATORIAS**:

a) Todas as clases deben incluír un **identificador persistente** (excepto as subclases da xerarquía, que o herdarán da superclase), **diferente da clave natural**, e con asignación de valores **automática** vía **@TableGenerator**.
Recomendación: usar o tipo *Long* para o identificador.

b) **Unha** das asociacións do modelo debe ser implementada como **bidireccional** (é dicir, definindo as propiedades/coleccións axeitadas nos dous extremos da asociación.

Ademais, debedes implementar os seguintes métodos nas clases:

MO2.1: Métodos ***equals*** e ***hashcode***, definidos usando a **clave natural** correspondente, **para CADA CLASE** do voso modelo. (NOTA: no caso da **xerarquía**, define estes métodos **unicamente** para a **superclase.** As subclases deben herdar os métodos definidos na superclase, sen ningunha modificación).

MO2.2: **DOUS** *"métodos de conveniencia"* **para (SOAMENTE) UNHA** das **asociacións** implementadas como **bidireccionais** no voso código, que actualicen **en memoria** os dous extremos da mesma **simultaneamente**: un para **crear** novas asociacións, e outro para **eliminar** asociacións xa existentes entre instancias das clases involucradas (Exemplo: ver métodos Usuario.engadirEntradaLog() e Usuario.eliminarEntradaLog() no código proporcionado)

.

3 “Mapeo” das clases de negocio

Definide os “mapeos” das clases do modelo utilizando **anotacións JPA**, tendo especial coidado cos seguintes aspectos:

a) Asegurádevos de que os tipos de datos das clases e a BD son compatibles e axeitados
b) Incluíde todas as anotacións necesarias para que Hibernate **cree automaticamente na BD** TODAS as restricións de unicidade e nulos que sexan pertinentes.
c) Incluíde as anotacións necesarias para que Hibernate utilice a estratexia de mapeo da **xerarquía** que consideredes mais axeitada.

Ademais, é **OBRIGATORIO** cumprir as seguintes condicións:

a) Configurade **como mínimo unha** das propiedades/coleccións que representen asociacións con **estratexia EAGER**.
b) Configurade **como mínimo unha** das propiedades/coleccións que representen asociacións con **estratexia LAZY**.
c) Configurade **como mínimo unha** das propiedades/coleccións que representen asociacións para que exista **propagación automática** de operacións. A operación a propagar pode ser *PERSIST*, *MERGE*, *REMOVE*. Elixide a que consideredes mais oportuna no contexto do voso dominio.

4 Implementación das clases *DAO*

Implementade unha ou varias clases *DAO* da mesma forma que na aplicación que se vos proporciona como exemplo. As clases *DAO* deben implementar os seguintes métodos:

MO4.1: Un método de **recuperación** por **clave natural**, usando unha consulta **estática** JPQL, **para CADA clase** (NOTA: no caso da **xerarquía**, un só método, para a **superclase**). Por exemplo, ver UsuarioDaoJPA.recuperaPorNif() no código de exemplo proporcionado.
MO4.2: Un método de **alta** de obxectos na BD **por CADA clase** do voso modelo (NOTA: no caso da **xerarquía**, un só método, para a **superclase**) utilizando *persist()*. Por exemplo, ver UsuarioDaoJPA.almacena() no código de exemplo proporcionado.
MO4.3: Un método de **eliminación** de obxectos da BD **por CADA clase** do voso modelo (NOTA: no caso da **xerarquía**, un só método, para a **superclase**) utilizando *remove()*. Por exemplo, ver UsuarioDaoJPA.elimina() no código de exemplo proporcionado.
MO4.4: Un método de **actualización** de obxectos da BD **por cada clase** do voso modelo (NOTA: no caso da **xerarquía**, un só método, para a **superclase**) utilizando *merge()*. Por exemplo, ver UsuarioDaoJPA.modifica() no código de exemplo proporcionado.
MO4.5: Un método que inicialice unha propiedade/colección LAZY fóra de sesión. Escollede **unha calquera** de entre todas as propiedades establecidas como LAZY **de entre todas as clases** do voso modelo (**só unha, en total**). O método debe recibir como argumento un obxecto coa propiedade LAZY en forma de *proxy* **sen inicializar**, e debe devolver o mesmo obxecto (ou unha copia del) coa propiedade xa **inicializada**. Por exemplo, ver o método UsuarioDaoJPA.restauraEntradasLog() no código de exemplo proporcionado. Ver tamén 5 ways to initialize lazy associations and when to use them (thorben-janssen.com).

MO4.6: **Catro métodos** que impliquen, cada un, unha consulta JPQL **que teña sentido** no dominio que lle corresponde ao teu diagrama. As consultas implementadas deben ser do seguinte tipo:

a) Unha consulta **dinámica** JPQL cun *inner join* entre dúas (ou mais) clases.
b) Unha consulta **dinámica** JPQL cun *outer join* entre dúas (ou mais) clases.
c) Unha consulta **dinámica** JPQL que inclúa unha subconsulta.
d) Unha consulta **dinámica** JPQL que utilice unha función de agregación (*count*, *avg*…)

AVISO IMPORTANTE:
-As consultas deben ser congruentes e non ser "forzadas". Por exemplo, unha consulta cun *JOIN* totalmente innecesario e que funcionaría exactamente igual sen el NON sería avaliada.
-É imprescindible que cada consulta JPQL que se pide (por exemplo en **MO4.1** ou **MO4.6**.) sexa implementada **exactamente** na forma indicada **(estática/dinámica).** NON serán avaliadas aquelas implementacións que non se correspondan estritamente co que se pide.
-Nas consultas que inclúan operacións de JOIN, está **EXPRESAMENTE PROHIBIDO** utilizar a **cláusula ON**. Tamén está prohibido establecer a condición de JOIN mediante unha cláusula WHERE. Usade notación de obxectos pura, como a explicada en clase.

5 Implementación dos tests de proba.

Debedes crear un conxunto de test *JUnit* para probar o correcto funcionamento da implementación realizada. Podedes usar como referencia os test definidos na aplicación que se vos proporciona como exemplo. A listaxe de tests a implementar é a seguinte:

Test01: Un test que probe **todos** os métodos de recuperación por clave natural (ver MO4.1) de instancias de **todas as clases** do voso modelo **(con excepción da superclase abstracta)**, para demostrar que funcionan correctamente.

Test02: Un test que probe todos os métodos de **alta** (ver MO4.2) de instancias de **todas as clases** do voso modelo **(con excepción da superclase abstracta)**, para demostrar que funcionan correctamente

Test03: Un test que probe todos os métodos de **eliminación** (ver MO4.3) de instancias de **todas as clases** do voso modelo **(con excepción da superclase abstracta)**:

1. Dando de alta na BD instancias temporais das ditas clases, usando os métodos DAO definidos en MO4.2.
2. Forzando a eliminación na BD das ditas instancias, usando os métodos DAO definidos en MO4.3.

Test04: Un test que probe todos os métodos de **modificación** (ver MO4.4) de instancias de **todas as clases** do voso modelo **(con excepción da superclase abstracta)**, para demostrar que funcionan correctamente.

1. Recuperándoas da BD, usando os métodos DAO definidos en MO4.1.
2. Modificando o seu estado básico en memoria

3. Forzando a gravación na BD dos cambios realizados, usando os métodos DAO definidos en MO4.4.

Test05: Un test que demostre que o método para forzar a inicialización da propiedade/colección LAZY (ver MO4.5) funciona correctamente. *(Exemplo: Supoñendo que a colección de logs da clase Usuario está configurada como LAZY: i) cargamos un usuario desde a BD; ii) comprobamos que a súa colección de logs non está inicializada aínda; iii) facemos unha chamada ao método a probar; e iv) demostramos que a propiedade xa foi inicializada.*

Test06: Un test que demostre que **unha calquera** (podedes elixir) das propiedades/coleccións que **representen asociacións** e estean configuradas **como EAGER** funciona correctamente. *(Exemplo: Supoñendo que a colección de logs da clase Usuario está configurada como EAGER, cargamos un usuario desde a BD e demostramos que a súa colección de logs foi xa inicializada automaticamente no momento da carga).*

Test07: Un test que demostre que unha calquera (podedes elixir) das configuracións de propagación automática de operacións activadas no voso modelo (ver "Mapeo" das clases de negocio) funciona correctamente.

Test08: Un test que comprobe que **todas** as consultas JPQL definidas (ver MO4.6) funcionan correctamente.