Practica 3. Aplicación de inventario: Gestión de usuarios.

1. Objetivos

La práctica consiste en el desarrollo de la parte de acceso a datos para la gestión de usuarios de la aplicación de inventario. Con la resolución de la práctica se contribuye a:

- Poner en práctica los conceptos de acceso a datos utilizando jdbc en el contexto de una aplicación real.
- Crear componentes de acceso a datos que hereden de clases de acceso a datos genéricas.
- Desarrollar aplicaciones separando la capa de acceso a datos del interfaz de usuario.

2. Introducción

En esta práctica desarrollaremos la parte de acceso a datos para la gestión de usuarios de la aplicación de inventario. En concreto, en esta práctica gestionaremos:

- la validación de usuarios en el sistema, es decir, comprobación de que un usuario entra en la aplicación con el nombre y contraseña correctos.
- el mantenimiento (alta, baja, modificación) de usuarios y de otras entidades relacionadas con usuarios.

Entidades implicadas

Si bien el objetivo final es la gestión de usuarios (tabla usuario de la BD), hay una otras entidades relacionadas directamente con ***usuarios***. Estas son:

- ***Departamentos***.
- ***Grupos***
- ***Tipos de usuarios***
- ***Roles***

En la práctica se hará también la gestión del acceso a los datos a estas entidades. En el punto 3 de este enunciado se dan más detalles sobre cada una de las tablas implicadas.

Uso del proyecto utiles

En la práctica se utilizará el proyecto *utiles* desarrollado en la práctica 1 para todo aquello relacionado con:

- Conexión y desconexión con la base de datos.
- Cierre de objetos (*PreparedStatement, ResultSet*, …)
- Generación de excepciones: Los métodos de la capa de acceso a datos lanzarán excepciones de la clase *BusinessException* del proyecto *utiles*

Objetos de acceso a datos y pojos.

La forma de **independizar** el interfaz de usuario del acceso a los datos consiste en utilizar una serie de clases que proporcionan al interfaz los métodos necesarios para realizar altas, modificaciones, eliminaciones y consultas de cada una de las entidades implicadas.

Los objetos de estas clases, denominados **objetos de acceso a datos (DAO)**, proporcionarán métodos para realizar las tareas necesarias. Por cada entidad existente en la aplicación implementaremos una clase de acceso a datos para dicha entidad. Así, por ejemplo, para manipular los datos de departamentos se implementará la clase DaoDepartamento, para los grupos, DaoGrupo, etc.

El interfaz de usuario no necesitará ser conocedor de cómo están almacenados los datos: si se guardan en ficheros de texto, en ficheros binarios o en una base de datos de determinado fabricante. Sólo necesitará conocer los métodos que proporcionan las clases DaoXXXXX.

Para posibilitar esta independencia entre la capa DAO y el interfaz, estas dos capas se comunican la información a través de POJO's.

POJO es el acrónimo de Plain Old Java Object (algo así como objeto simple de Java). Un pojo es una clase que representa a una de las entidades de la aplicación y tiene atributos para cada uno de los datos relevantes de dicha entidad y los métodos getter y setter correspondientes. Así, por ejemplo, el pojo *Departamento* dispondrá de atributos para almacenar el id y el nombre de un departamento y métodos para modificarlos y consultarlos.

Si el interfaz de usuario llama a un método de la capa de acceso a datos (capa dao) y necesita pasarle información de un departamento, lo hará pasándole la información en un objeto (pojo) de la clase Departamento como parámetro. Del mismo modo, si un método de la capa dao va a proporcionar información de un departamento al interfaz de usuario, lo hará devolviendo un objeto de la clase Departamento.

Clases genéricas de acceso a datos.

Durante el desarrollo de la práctica tendrás la oportunidad de comprobar que muchas de las tareas que se hacen en los métodos de la capa dao son repetitivas. Se realizan de forma casi idéntica en las distintas clases de la capa: En DaoDepartamento, en DaoGrupo, etc.

Resulta útil, y es posible, reducir esta "duplicidad" de código. La solución pasa por hacer que las clases DaoXXXXX hereden de una clase común (a la que llamaremos **DaoGenerico**). La clase DaoGenerico contendría el código que no cambia de unas clases DaoXXXXX a otras, y en las clases DaoXXXXX se implementarían los métodos específicos de la entidad en cuestión.

Por ejemplo, todas las clases DaoXXXXX son susceptibles de tener un método para eliminar una instancia de dicha entidad (DaoDepartamento para borrar un departamento, DaoGrupo para borrar un grupo, …). El método eliminar se podría implementar en DaoGenerico.

Sin embargo, la validación es algo que atañe únicamente a la entidad usuario y por tanto no tendría sentido implementar el método que valida a un usuario en la clase DaoGenerico (puesto que no es un método común a todas las entidades). El método validar estaría implementado en la clase DaoUsuario únicamente.

En esta práctica se sientan las bases para hacer posible esta reducción de código pero, dado que no es posible atender a todos los frentes en una sola práctica, la reutilización de código se deja para prácticas posteriores y por el momento habrá código muy parecido en las distintas clases de la capa de acceso a datos.

Aun así, las clases DaoXXXXX heredarán de la clase DaoGenerico, quien a su vez implementa *InterfazDaoGenerico.* La clase DaoGenerico (por el momento) estará prácticamente vacía. Ya centraremos la atención en la reutilización de código en el futuro.

3. La base de datos.

La base de datos está compuesta por numerosas tablas. De momento trabajaremos con cinco de ellas: ROL, DEPARTAMENTO, GRUPO, TIPOUSUARIO y USUARIO. Se describen a continuación.

La tabla departamento.

Contiene los departamentos didácticos que existen en el centro. Los profesores pertenecen a algún departamento.

La tabla grupo.

Contiene los grupos de alumnos que hay en el centro (1º ESO A, 1º DAM, etc). Los alumnos pertenecen a algún grupo. Observa que **el id** de las filas de esta tabla **no es numérico**.

La tabla rol.

Contiene los posibles roles que puede desempeñar un usuario de la aplicación (administrador, directivo, usuario normal, etc). El rol de un usuario determina qué cosas puede hacer en la aplicación y cuáles no. Los roles están relacionados, a su vez, con permisos, pero por el momento no entraremos en la gestión de los permisos que tiene el usuario.

La tabla tipousuario.

Un usuario puede ser de tres tipos: (1, “profesor”), (2, “alumno”) o (3, “pas”) (personal de administración y servicios). La aplicación de inventario tratará de forma distinta a los usuarios dependiendo del tipo que sean.

Consideraremos el **contenido** de esta tabla como **fijo** y la aplicación no tendrá los correspondientes mantenimientos (altas, eliminaciones, …) de tipos de usuario. En consecuencia, la clase DaoTipoUsuario únicamente implementará aquellos métodos relacionados con consultas de información y dejará sin sobreescribir los que suponen la edición de tipos de usuarios. **(1)**

La tabla usuario.

Esta es la tabla más compleja de entre las nombradas hasta ahora. Contiene la información de los usuarios de la aplicación: datos personales, nombre de usuario y contraseña para entrar a la aplicación, etc. Además contiene los campos que relacionan esta tabla con rol, grupo y departamento (claves ajenas).

A la hora de añadir y modificar usuarios habrá que asegurar que se cumplen las siguientes restricciones que no están implementadas a nivel de base de datos:

- Los usuarios de tipo *profesor* tienen que estar relacionados con un departamento y no estar relacionados con ningún grupo. Es decir, que si el tipousuario es profesor, el campo departamento debe ser distinto de null, mientras que el campo grupo será necesariamente null.
- Con los usuarios de tipo *alumno* ocurre justamente al contrario: Estarán relacionados con un grupo y con ningún departamento.
- Los usuarios de tipo pas no pueden estar relacionados con ningún grupo ni departamento.

En consecuencia, en la clase DaoUsuario:

- Al añadir un usuario, se comprobarán las restricciones anteriores y se lanzará BusinessException en caso de que alguna no se cumpla.
- Al modificar, si el usuario cambia de tipo, se hará de forma que se cumplan las restricciones anteriores. Por ejemplo, si el usuario pasa a ser *profesor*, el campo grupo se pondrá a null. Surgirá por tanto la necesidad de poner a null ciertos campos de la base de datos. En clase no hemos visto como hacerlo, así que tendrás que buscar información de cómo conseguirlo. **(2)**

4. Trabajo a realizar.

Los pasos que debes seguir para completar la práctica son los siguientes:

1. Crea el proyecto *inventario* y estructúralo convenientemente.
   1.1. Crea los paquetes *dao*, *pojos* e *interfaz.*
   1.2. Crea el folder configuración y añade un fichero de configuración para poder conectar con la base de datos.
   1.3. Añade al proyecto una referencia al proyecto *utiles* (como ya se hizo con testDeUtiles en la práctica 1): Propiedades del proyecto – Java Build Path – Pestaña projects.
2. En el proyecto utiles crea un paquete llamado *dao* y añade en él la clase DaoGenerico y el interfaz *interfazDaoGenerico* que se te han proporcionado junto al enunciado.
3. En el paquete pojos, crea un pojo para cada entidad relacionada con la gestión de usuarios: Departamento, Grupo, Rol, TipoUsuario, Usuario. Cada uno de los pojo:
   3.1. Definirá los atributos necesarios (privados): uno por cada campo de la tabla a la que representan. NO usaremos tipos primitivos para los atributos, sino clases (Integer, Double, etc).
   3.2. Implementará un constructor vacío.
   3.3. Implementará un constructor que reciba todos los datos del objeto. Los recibirá en el mismo orden en que aparecen los campos correspondientes en la base de datos. Esto último no es un requisito pero es una manera de que todos lo hagamos igual.
   3.4. Implementará métodos getter y setter de todos los atributos.

3.5. Implementará los métodos equals, hashCode y compareTo. En los tres casos el criterio de comparación y de generación del código hash estará basado en el identificador de la entidad en cuestión (id de departamento en Departamento, id de grupo en Grupo, etc). Os recuerdo que Eclipse genera automáticamente código de equals() y hashCode()

3.6. Implementará el método toString de manera que devuelva una cadena con todos los datos de la entidad en la misma línea.

4. En el paquete *dao* implementar las clases DaoDepartamento, DaoGrupo, DaoRol, DaoTipoUsuario y DaoUsuario.
   4.1. Las clases heredarán de DaoGenerico e implementarán InterfazDaoGenerico.
   4.2. Todos los métodos desarrollados asumirán que la conexión con la base de datos ya está establecida y se limitarán a obtenerla y utilizarla, usando para ello el método ConexionJdbc.getConnection(). La conexión se realizará y se cerrará desde el interfaz de usuario.
   4.3. Cada una de las clases (excepto DaoTipoUsuario) sobreescribirá todos los métodos de DaoGenerico para que realicen la tarea que se describe en la documentación correspondiente (ver comentarios en la cabecera de los métodos de InterfazDaoGenerico).
   4.4. La clase DaoTipoUsuario sólo implementará los métodos que no supongan inserción, eliminación o modificación de un tipousuario. Anteriormente **(1)** se ha explicado el motivo. El resto de métodos se dejará sin implementar. Si por error se llamara a alguno de los métodos no implementados, se ejecutaría el código del método existente en la clase DaoGenérico que, como podrás observar, lanza siempre una RuntimeException.
   4.5. En la clase DaoTipoUsuario se implementarán tres **constantes** enteras: PROFESOR = 1, ALUMNO = 2 y PAS = 3. Estas constantes representan los tres tipos de usuarios existentes.
   4.6. La clase DaoUsuario se implementará teniendo en cuenta las restricciones que hemos explicado anteriormente **(2)**
   4.7. La clase DaoUsuario tendrá, además, un método validar, que reciba usuario y contraseña y devuelva un boolean indicando si son correctos o no.
5. Realización de algunas pruebas: En el paquete *interfaz* del proyecto se implementarán programas para realizar algunas pruebas de los métodos de la capa dao.
   5.1. (clase TestAñadirDepartamento): Solicitará al usuario los datos de un departamento y lo añadirá a la base de datos llamando al método correspondiente de la capa DAO. Si el alta se realiza se mostrará la información del nuevo registro. Si el alta falla se mostrará un mensaje de error.
   5.2. (clase TestValidarUsuario): Se solicitará al usuario nombre de usuario y contraseña y se le indicará si puede entrar al sistema o no.
   5.3. (clase TestConsultarUsuario): Se solicitará al usuario un id de usuario. Si el usuario no existe se mostrará un mensaje por pantalla indicándolo. Si el usuario existe se mostrará su información por pantalla y:
      - Si se trata de un PROFESOR se mostrará el nombre del departamento al que pertenece.
      - Si se trata de un ALUMNO se mostrará el nombre del grupo al que pertenece.

5.4. (clase TestConsultarDepartamentos): Se solicitará al usuario un nombre de departamento. Si el departamento existe se mostrará por pantalla su id y si no existe, un mensaje indicándolo.

5. Entrega de la práctica.

- Se entregará únicamente el proyecto inventario, puesto que el proyecto útiles es común y no hay que realizar cambios en el.
- Fecha de entrega: Hasta el lunes 27 de octubre a las 23:55