UNIVERSIDAD DE LA EMPRESA

Bases de Datos III
Práctico 4
Licenciatura en Informática
Ingeniería en Informática

Ejercicio 1

Para cada una de las siguientes afirmaciones sobre aplicaciones en ***arquitecturas de 2 capas***, indique si es correcta o incorrecta de acuerdo a los conceptos vistos en el teórico. Fundamente todas sus respuestas.

a) Si una aplicación utiliza el patrón **MVC**, pero ***no*** utiliza los patrones **Facade** y **Value Object**, entonces sigue siendo una aplicación en 2 capas.

b) Si una aplicación utiliza los patrones **Facade** y **Value Object** pero ***no*** utiliza el patrón **MVC**, entonces sigue siendo una aplicación en 2 capas.

c) Si una aplicación utiliza el patrón **Facade** pero ***no*** utiliza los patrones **MVC** y **Value Object**, entonces sigue siendo una aplicación en 2 capas.

d) Toda aplicación en 2 capas necesariamente debe hacer uso de ***stored procedures*** para que se la considere una aplicación en 2 capas.

e) Una aplicación que hace uso de un **Pool de Conexiones**, es necesariamente una aplicación en 2 capas y no una aplicación en 1 capa.

f) Toda aplicación en 2 capas necesariamente posee una arquitectura física ***distribuida***.

Ejercicio 2

Se va a desarrollar una ***aplicación en 2 capas*** que permitirá resolver requerimientos de una sala de juegos infantiles. La aplicación correrá en una LAN y tendrá una ***arquitectura física 3-Tier*** como la mostrada en la figura:

Habrá un servidor central en el que residirá una fachada que resolverá los requerimientos y accederá a la base de datos, que residirá en un equipo separado. Los usuarios interactuarán con la aplicación mediante una ***interfaz gráfica de ventanas***, la cual será accedida mediante **RMI** desde los equipos clientes.

El esquema de la base de datos **MySQL** para la aplicación será el siguiente:

```
Niños (cédula INT, nombre VARCHAR(45), apellido VARCHAR(45))
Juguetes (número INT, descripción VARCHAR(45), cédulaNiño INT)
```

- La columna `cédulaNiño` en `Juguetes` referencia al campo `cédula` en `Niños`.
- Puede ocurrir que niños diferentes tengan juguetes distintos con el mismo número, por esta razón ninguna de las columnas de `Juguetes` fue marcada expresamente como clave primaria.

Escriba un programa `Main` en **Java** que cree la base de datos según el esquema descrito y cargue en ella los datos de los siguientes niños por defecto:

cédula	nombre	Apellido
1234567	Kevin	McCallister
2345678	Matilda	Wormwood
3456789	Harry	Potter
4567890	Merlina	Adams

Ejercicio 3

En este ejercicio desarrollaremos la aplicación descrita en el ejercicio 2. Por el momento supondremos que existirá un único usuario y **no** nos preocuparemos de realizar manejo de transacciones. Las siguientes clases en UML corresponden a la fachada de acceso a la capa lógica y de persistencia y a los value objects a utilizar para transferir información entre las dos capas:

<< logica / persistencia >> FACHADA
- atributos para conexión BD
+ Fachada ( ) + nuevoNiño (voN: VONiño) : void + nuevoJuguete (desc: String, cedN: int) : void + listarNiños ( ) : List <VONiño> + listarJuguetes (cedN: int) : List <VOJuguete> + darDescripción (cedN: int, numJ: int) : String + borrarNiñoJuguetes (cedN: int) : void

<< grafica / logica >> VONIÑO
- cédula : int - nombre : String - apellido : String
+ VONiño (int, String, String) + getCedula ( ) : int + getNombre ( ) : String + getApellido ( ) : String

<< grafica / logica >> VOJUGUETE
- número : int - descripción : String - cédulaNiño : int
+ VOJuguete (int, String, int) + getNumero ( ) : int + getDescripcion ( ) : String + getCedulaNiño ( ) : int

- El método `nuevoNiño` ingresa un nuevo niño al sistema, chequeando que no existiera.
- El método `nuevoJuguete` ingresa un nuevo juguete al sistema, chequeando que el niño que le corresponde esté registrado. El programa asignará automáticamente al nuevo juguete el número siguiente al del último juguete que ya tenía el niño. Por ejemplo, si tenía 5 juguetes, asignará el nº 6.
- El método `listarNiños` devuelve un listado de todos los niños registrados, ordenado por cédula.
- El método `listarJuguetes` devuelve un listado de todos los juguetes de un niño determinado, (chequeando que dicho niño esté registrado) ordenado por número de juguete.
- El método `darDescripción` devuelve la descripción de un juguete, dados su número y la cédula del niño que le corresponde (chequeando que el niño exista y tenga un juguete con ese número).
- El método `borrarNiñoJuguetes` elimina del sistema al niño con la cédula ingresada, y también elimina a todos sus juguetes, chequeando que el niño esté registrado.

Observación: En caso de que cualquiera de los chequeos mencionados falle, la fachada lanzará una ***excepción personalizada*** para notificar el correspondiente error a la capa gráfica.

a) Cree un proyecto para la aplicación con la estructura de packages dada en la figura.

b) Implemente en el package `excepciones` todas las clases correspondientes a las posibles excepciones que puedan ocurrir.

c) Implemente en el package `valueObjects` las clases `VONiño` y `VOJuguete` correspondientes a los value objects especificados en UML.

d) Implemente en el package `accesoBD` la clase `Consultas` que define los textos de todas las sentencias SQL que la aplicación necesite ejecutar sobre la base de datos.

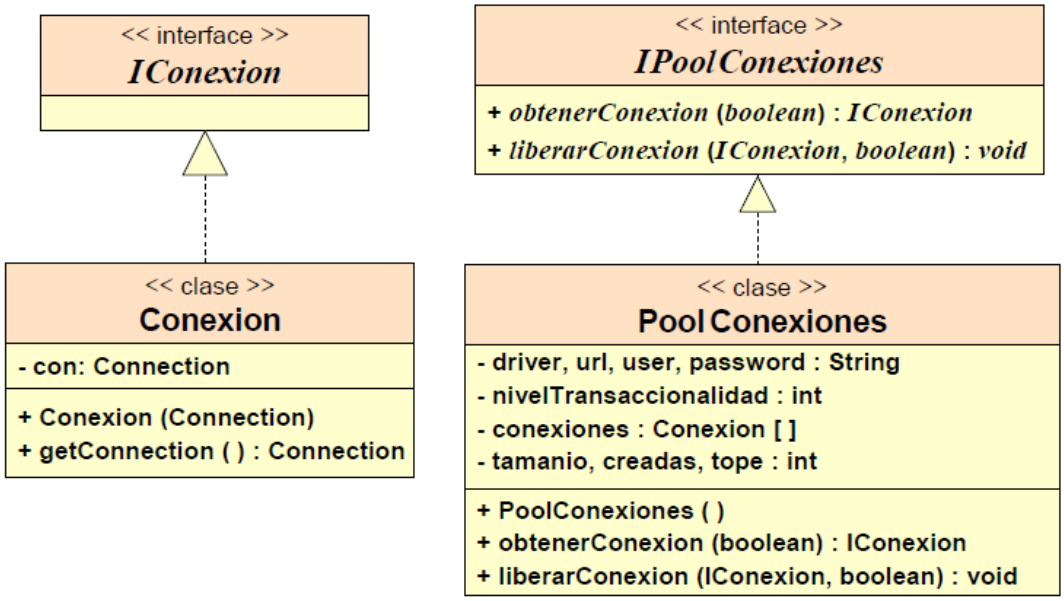
e) Implemente en el package `accesoBD` la clase `AccesoBD` que encapsula el acceso a la BD. Debe poseer los métodos que ejecuten las sentencias SQL definidas en la clase `Consultas`.

f) Implemente en el package `logicaPersistencia` la clase `Fachada` de modo que sus métodos se comporten según la descripción dada antes. Hará uso de la clase `AccesoBD` y será accedida desde los equipos clientes mediante **RMI** (Remote Method Invocation).

g) Implemente en los packages `ventanas` y `controladores` las clases correspondientes a la capa gráfica de la aplicación. Las mismas deberán aplicar el patrón **Model View Controller** y brindar las ventanas necesarias para dar solución a los requerimientos de la aplicación. Los controladores accederán a la Fachada mediante **RMI** (Remote Method Invocation).

Ejercicio 4

En este ejercicio desarrollaremos en **Java** un **Pool de Conexiones** como el visto en el teórico. La implementación a realizar en este práctico utilizará internamente una estructura de arreglo con tope para albergar las conexiones. Luego haremos uso del pool para permitir el acceso de usuarios en forma ***concurrente*** y realizar ***manejo de transacciones*** en la aplicación del ejercicio anterior.



- La descripción de los atributos y métodos de cada una de las clases e interfaces es la que está dada en el capítulo 4 del material teórico.
- Los métodos `obtenerConexion` y `liberarConexion` internamente harán uso de las primitivas `wait` y `notify` de **Java** para manejo de concurrencia.
- Las conexiones irán siendo instanciadas **a demanda**. Es decir, sólo se creará una nueva conexión cuando todas estén actualmente en uso y aún haya espacio para crear nuevas.
- La carga de `driver`, `url`, `user` y `password` de la base de datos será realizada en el método constructor del Pool desde un ***archivo de configuración***.
- Los métodos de la clase del Pool lanzarán una `PersistenciaException` como **única** excepción en caso de ocurrir cualquier error de comunicación con la BD.

a) En la estructura de packages del ejercicio anterior, cree un package `poolConexiones` dentro de la capa lógica/persistencia. Implemente en dicho package las clases e interfaces anteriores con **todos** sus métodos. Luego Haga un programa `main` de prueba que permita testear el correcto funcionamiento del Pool de Conexiones.

b) Incorpore el uso del Pool de Conexiones a la aplicación del ejercicio anterior. Para ello, incorpore a la Fachada el siguiente atributo: `private IPoolConexiones pool;` Luego modifique la implementación de cada requerimiento de modo que:

- Lo primero que haga el requerimiento sea solicitar una *`IConexion`* al pool.
- Mantenga exactamente el mismo comportamiento que ya tenía, sólo que la conexión usada ahora será la obtenida del pool en vez de la conexión aislada usada antes. Dentro de la clase `AccesoBD` deberá ahora castear la *`IConexion`* hacia una `Conexion` concreta.
- Devuelva la *`IConexion`* al pool, finalizando internamente la transacción mediante `commit` o `rollback`, según corresponda.