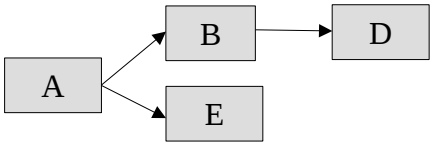
PRÁCTICA 5 de Java: Final (11/11/2023):

Bases de datos

Fecha tope entrega	Lunes, 25 de noviembre.	**Fecha tope defensa**	Martes, 26 de noviembre.
Tipo	Individual	**Entrega**	Github Classroom



Requisitos:

1.- Base de datos: desarrolla una BDs. que sea lo más real posible, a proponer por el alumno y con el visto bueno del profesor. El sistema de BDs (MySQL, MariaDB, PostgeSQL, SQLite, etc.) ejecutandose sobre Docker y tendrá las siguientes características:

1.1.- Similar al esquema ER de arriba.

1.2.- Una tabla de validación (A): algo como socio, cliente, usuario, etc. Para validarse en la aplicación con los campos **usuario** y **contraseña**.

1.3.- En A habrá una imagen, un campo calculado sobre valores de E y campo tope (máximo o mínimo)

1.4.- Existirán campos de tipo entero, real, alfanumérico y fecha.

1.5.- Existirá el campo NIF.

1.6.- Nomenclatura FK: tendrán las tres primeras letras del nombre de la tabla de donde es PK y el resto el nombre del atributo PK. Por ejemplo la FK cuenta.cliNumero procede de la PK cliente.numero

2.- Vista. Desarrolla una aplicación gráfica con las siguientes características:

2.1.- Los JFrame y los JPanel deberán ser clases en ficheros independientes.

2.2,. Inicialmente no se podrá acceder a la aplicación o los diferentes opciones estarán deshabilitadas, salvo para iniciar sesión y salir.

2.3.- Inicio de sesión: consistirá en encontrar una única fila de la tabla de validación A (socio, cliente, etc.) mediante el usuario y la contraseña. Esto es independiente de la conexión a la BDs con el usuario administrador, aunque se aconsejan que se hagan juntas

2.3.1.- Permitirá acceder a la aplicación o habilitará todas las funciones de la aplicación (menús).

2.4.- Cerrar sesión. Deshabilita el uso de la aplicación.

2.5.- Diferentes paneles de gestión entre los que habrá:

2.5.1.- Ver filas una a una de la tabla B (*sensitive* y *updatable*) con diversas acciones con botones:

Se recomienda isFirst(), isLast(), rs.updateRow(), deleteRow(), conn.setAutoCommit(false), conn.commit(), etc.
https://docs.oracle.com/en/java/javase/17/docs/api/java.sql/java/sql/ResultSet.html

- Avanzar, retroceder, primero y último
- Modificar y borrar el elemento visualizado (cuidado si está relacionado con C):
- En una ventana modal mostrar en un Jlist los elementos de la tabla C relacionados.

2.5.2.- Ver filas en un Jtable (tabla E).

2.5.3.- Ver perfil del usuario validado (tabla A).

2.5.4.- Modificar foto tabla A.

2.5.5.- Altas en la tabla E.

2.5.6.- Bajas en la tabla E.

2.5.7.- El campo calculado de la tabla se verá afectado por las altas y bajas en E. Hay que controlar y alertar mediante excepciones si se sobrepasa el valor del campo tope.

2.6.- Otros:

2.6.1.- Se permitirá modificar la imagen, una fecha y el NIF.

2.6.2.- Las imágenes estarán en una carpeta pero se podrán buscar en cualquier lugar mediante un filechooser, para luego copiarla a dicha carpeta.

2.6.3.- La fecha, se podrá modificar usando DatePicker o DataChooser. Deberá haber la posibilidad de errores en la fecha controlados por el sistemas de errores.

2.6.4.- El NIF será validado mediante la librería adjunta ValNif08201.zip de la Agencia Tributaria.

2.7.- Habrá un JPanel “Acerca de” con la información del autor, versión, etc..

3.- Modelo:

3.1.- Habrá una clase por cada tabla de la base de datos. POJOs (Plain Old Java Objects).

3.2.- Para las tablas A, C y E existirá una clase por cada table donde están las consultas y las llamadas a las clase controlador para obtener los datos en forma de collections u objetos (DAO - Data Access Object)

- Llevarán por parámetros la consulta y los campos que la cumplimentan (Statement o PreparedStatement) a las clases que gestionan la BDs.

- Devolverán una collection de objetos o un objeto, según el tipo de consulta.

4.- Controlador. Estará compuesto de varias clases:

4.1.- La clase para la gestión de la conexión.

- La validación usará un PreparedStatement.

4.2.- Una clase (o una por tabla) para la gestión de las operaciones select, insert, delete y update. Sus métodos reciben una cadena y devuelven un objeto o una collection de objetos. En los insert o update, reciben una cadena y devuelven un entero con el número de filas afectadas.

- Para las consultas repetitivas (Jlist del uno a uno) se usará un PreparedStatement.

https://www.arquitecturajava.com/jdbc-prepared-statement-y-su-manejo/

https://docs.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.html

4.3.- Clase especial para mostrar uno a uno las filas. Tendrá, al menos, los métodos necesarios: iniciar(sql), avanzar(), retroceder(), irPrimero(), irUltimo(), esPrimero(), esUltimo, modificar(), borrar() y finalizar(). Además del método “objeto leer()”.

5.- Sistemas de Errores (dentro del controlador). Pretende diferenciar los errores para el usuario de la aplicación (por ventana) y los del sistema (Log)

5.1.- Mensajes de error: habrá una clase donde estén todos los errores informativos para el usuario de la aplicación: será estática, gestionará “todos” los errores de forma preestablecida y se usará para informar al usuario de la aplicación.

- Consistirá en una estructura de objetos con el numero de error y mensaje de error asociado.

- Tendrá un método con un número por parámetro y devolverá el mensaje asociado y el número. PE: “Error 1: no hay conexión a la base de datos”

5.2.- Excepción personalizada (hereda de Exception): será una clase para propagar o subir los errores/excepciones del sistema a la vista, y des allí gestionarlos con la clase “mensajes de error” (apartado 5.1).

- Hay que añadirle el atributo número de error con los métodos get y set.

5.3.- Gestión de log. Se guardará en un fichero el mensaje original (Exception) de los errores producidos, junto con la hora y la fecha

5.4.- Uso o funcionamiento de las clases anteriores:

- En los try-catch del sistema se realizarán dos acciones en los catch:

- Guardar la fecha, hora y el mensaje original del sistema en el fichero log.

Nota: En la fase de desarrollo se aconseja imprimir los errores a consola.

- Lanzar una excepción personalizada (5.2) con el número de error establecido (5.1).

- Si el error está controlado por un “if” u otra instrucción, se procederá de igual modo perro dentro del if: registrar en log y lanzar la excepción personalizada con el número de error.

6.- Exigencias del MVC:

6.1.- Será en la vista donde muestren los mensaje (JOptionPane, JDialog, etc) informativos o de error, con la ayuda de la Clase para los mensajes de errore (5.1). No se podrán llamar desde otros lugar.

6.2.- Independencia de clases, sobre todo por capas del MVC. Las clases se comunicarán con otras clase a través de métodos y no podrán acceder directamente a los atributos. PE: No se puede usar directamente un ResultSet en un JPanel.

6.3.- Cada parte del MVC tendrá su propio paquete.

7.- Las librerías externas que se añadan, estarán en la carpeta “lib” del proyecto.

Ayuda:

Se adjunto dos paquetes: uno como ejemplo del MVC y la gestión de errores. Y otro con hilos.

Ayuda Docker:

OpenWebinars:

Introducción a Docker:

https://openwebinars.net/cursos/docker-introduccion/

- Con MySQL
    - https://hub.docker.com/_/mysql
    - https://www.youtube.com/watch?v=kphq2TsVRIs
- Con MariaDB:
    - https://hub.docker.com/_/mariadb
    - https://mariadb.com/kb/en/installing-and-using-mariadb-via-docker/
- Con PostgreSQL
    - https://hub.docker.com/_/postgres
    - https://www.youtube.com/watch?v=hVrKX2RtigQ