Repaso examen 1Eva

ORDEN DE CREACIÓN NUEVO PROYECTO

1. Quitar el module-info y añadir VM model
    a. Añadir la clase ejecutable
    b. Añadir el VM Model: --module-path "C:\Archivos de programa\Java\javafx-sdk-23\lib" --add-modules javafx.controls,javafx.fxml
    c. Eliminar el module-info
2. Leer bien el enunciado y dividir los directorios y carpetas con sus respectivas clases.
3. Configurar el Scene Builder
    a. Buscarlo en el buscador
    b. Abri su ubicación
    c. Copiar la ruta
    d. Elegir el .exe

BUSCAR INFO DE CLASE

1_Encriptacion_sha256

- Encriptado de contraseñas

4_CRUD_Hibernate_cfg_usuarios_AutoIncrement_BD

- CRUD básico con hibernate. Incluye el insertar, borrar, actualizar, buscar por id y listar usuarios.

4_CRUD_HibernateUsuarios_modificado

- Utiliza el formateo de fechas de LocalDate.

4_HibernateRelacion1aN

- Resumen del 1 a N sin clase DAO, todo desde la App.
- Bien descrito lo que hay que poner en cada campo del OnetoMany y el ManytoOne.

Cuentas_Bancarias

- Uso de polimorfismo.
- Uso de clases abstractas.

BUSCAR INFO DE ENTREGAS

Entrega 1 – Periódico Online con JavaFX

- Clase abstracta
- RadioButton con ToggleGroup

Entrega 2 – Tienda de cosmética con JavaFX y SQL

- Uso de imágenes en FX
- RadioButton con ToggleGroup
- Convertir LocalDate a SQL (clase VentasDAO)
- Uso de SQL
    - Resources → Configuration → database.properties:
      host=localhost
      port=3306
      name=NOMBRE_BBDD
      username=root
      password=toor
    - Java → Util → ConexionBBDD:

```
public class conexionBBDD {
    public static Connection conectar() {
        Connection connection = null;
        try {
            String url =
"jdbc:mysql://localhost:3306/NOMBRE_TABLA?serverTimezone=E
urope/Madrid"; //me daba error: "problema con la
configuración de la zona horaria en la conexión de tu base
de datos MySQL", por lo que pongo
"?serverTimezone=Europe/Madrid"
            String user = "root";
            String password = "toor";
            connection = DriverManager.getConnection(url,
user, password);
        } catch (SQLException e) {
            System.out.println("No se ha establecido la
conexión con la base de datos.  " + e.getMessage());
        }
        return connection;
    }//conectar
```

Entrega 3 – Peliculas: fichero JSON y ListView

- JSON en My SQL Workbench
- ListView

***En el ListView aparecen solo los títulos porque es lo que marca el toString de la clase Pelicula (Model → Pelicula).**

Entrega 4 - CRUD Coches con Mongo y TableView

- Uso de TableView
- Uso de MongoDB: **collection = tabla y document = fila**
    - Resources → Configuration → database.properties:

```
host=localhost
port=27017
author= NOMBRE_BBDD
username=admin
password=1234
```

    - Java → Util → ConexionBBDD:

```
import com.mongodb.MongoClient;
import com.mongodb.MongoClientURI;

import java.io.FileInputStream;
import java.util.Properties;

public class ConexionBBDD {

   public static MongoClient conectar() {
      MongoClient con;

      Properties properties = new Properties();
      String host = "";
      String port = "";
      String name = "";
      String username = "";
      String password = "";

      try {
         properties.load(new
FileInputStream("src/main/resources/Configuration/database.prope
rties"));
         host = String.valueOf(properties.get("host"));
         port = String.valueOf(properties.get("port"));
         name = String.valueOf(properties.get("name"));
         username = String.valueOf(properties.get("username"));
         password = String.valueOf(properties.get("password"));
         System.out.println(host + "  " + port + "  " + name + "  " +
username + "  " + password);
         con = new MongoClient(new MongoClientURI("mongodb://" +
username + ":" + password + "@" + host + ":" + port +
"/?authSource=admin"));
         return con;
      } catch (Exception e) {
         System.out.println("Conexion Fallida");
         System.out.println(e.getMessage());
         return null;
      }//try-catch
   }//conectar

   public static void desconectar(MongoClient con) {
      con.close();
   }//desconectar
```

ENTREGA 5 – CRUD HIBERNATE TUTORIAL ; USUARIO Y PROFESOR

- Uso de Hibernate:

- Resources → Configuration → hibernate.cfg.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-
configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- configuración de la conexión -->
        <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc
.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:330
6/NOMBRE_BASEDATOS</property>
        <property
name="hibernate.connection.username">root</property>
        <property
name="hibernate.connection.password">toor</property>

        <!-- dialecto de la base de datos -->
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQL8Diale
ct</property>

        <!-- mostrar consultas SQL en la consola -->
        <property
name="hibernate.show_sql">true</property>

        <!-- configuración de actualización de esquema -->
        <property
name="hibernate.hbm2ddl.auto">update</property>
    </session-factory>
</hibernate-configuration>
```

- Java → Util → HibernateUtil

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    //crea instancias de Session, que son las que ejecutan
las operaciones CRUD en la BD
    //usaré SessionFactory para abrir Session cuando
necesite realizar operaciones con la base de datos
    static SessionFactory factory = null; //variable
estática para el SessionFactory

    static {
        try {
            //cargar la configuración desde el archivo
hibernate.cfg.xml
            Configuration cfg = new Configuration();
//instancia de la clase Configuration de Hibernate, que
representa la configuración necesaria para establecer una
conexión con la base de datos y gestionar el mapeo entre
las entidades y las tablas

cfg.configure("Configuration/hibernate.cfg.xml"); //carga
la configuración desde el archivo hibernate.cfg.xml, que
contiene los parámetros necesarios para la conexión a la
BD
```

```
            cfg.addAnnotatedClass(Coche.class); //registra
la clase coche como entidad
            factory = cfg.buildSessionFactory();
//construye el sessionfactory
        } catch (Throwable ex) {
            //si algo sale mal, lanzar una excepción
            throw new ExceptionInInitializerError(ex);
        }
    }

    //método para obtener el sessionfactory
    public static SessionFactory getSessionFactory() {
        return factory;
    }

    //método para abrir una nueva sesión
    public static Session getSession() {
        return factory.openSession();
    }
}//class
```

Entrega 6 – CRUD Coches con Hibernate

- Uso de Interface
- ComboBox
- TableView

Entrega 7 – CRUD Coches Hibernate 1 a N

- En las clases DAO:
  - Poner al principio: SessionFactory factory = HibernateUtil.getSessionFactory();
  - En el try de las consultas, poner Session sesión = factory.openSession();
- DatePicker
- Cambio de escena pasando controller

Entrega 7 – Bueno_CRUD Coches Hibernate 1 a N

- Entrega de Adrián, pero todo con comentarios.
- Repasar el proyecto 4_HibernateRelacion1aN porque tiene todo detallado del MAnytoOne y el OnetoMany.

AYUDAS

ALERTA

//creo un método para las alertas que quiera introducir en mi programa --> evito demasiado código

```
public class Alertas {
   public static void mostrarAlerta(String mensaje, Alert.AlertType tipo) {
      Alert alert = new Alert(tipo); // utilizo el tipo de alerta pasado como parámetro
      alert.setTitle("Total.");
```

```
    alert.setContentText(mensaje); //mensaje que muestra cuando salta la alerta, que paso como parámetro
    alert.showAndWait(); //si no se cierra la ventana, no me permite continuar
  }//Alertas
```

SALIR DEL PROGRAMA

```
//función para salir del programa
public static void salir() {
  int opcion = JOptionPane.showConfirmDialog(null,
      "¿Está seguro de que desea salir del prgrama?", "Confirmación",
JOptionPane.YES_NO_OPTION);
  if (opcion == JOptionPane.YES_OPTION) { //Si la opción elegida es "Sí" se cierra
    System.exit(0); //Cierro la aplicación
  }
}//salir
```

CAMBIAR DE ESCENA SIN PASAR CONTROLLERS

```
// función para cambiar la escena a un nuevo FXML
public static void cambiarEscena(Button boton, String fxmlFile) {
  try {
    FXMLLoader fxmlLoader = new
FXMLLoader(HelloApplication.class.getResource(fxmlFile)); // Verifica la ruta correcta de Compras.fxml --> obtengo un controlador
    Parent root = fxmlLoader.load(); // Carga el archivo FXML
    Scene scene = new Scene(root); // Crea una nueva escena con el archivo FXML cargado
    Stage stage = (Stage) boton.getScene().getWindow(); // Obtén la ventana (Stage) desde el botón
    stage.setScene(scene); // Establece la nueva escena en la ventana actual
  } catch (Exception e) {
    System.out.println("Error al cambiar la escena." + e.getMessage());
  }//catch
}//cambiarEscena
```

PASAR DE ESCENA PASANDO CONTROLLER

```
try {
    FXMLLoader fxmlLoader = new
FXMLLoader(App.class.getResource("multas.fxml"));
    Parent root = fxmlLoader.load();

    //objetivo de lo siguiente --> pasar un coche (el seleccionado en esta pantalla) a la clase MultasController
    MultasController multasController = fxmlLoader.getController(); //cojo los datos de la clase MultasController (todos los datos de esa clase, o sea que puedo llamar a cualquier método que pueda necesitar)
    //con el control total de MultasController, llamo al método datosCocheMulta para pasarle el coche seleccionado
```

```
      multasController.datosCocheMulta(cocheSeleccionado); //le paso el coche
seleccionado al método de la clase MultasController

      Scene scene = new Scene(root);
      Stage stage = (Stage) verMultasBoton.getScene().getWindow();
      stage.setScene(scene);
   } catch (Exception e) {
      System.out.println(e.getMessage());
   }//try-catch
}//if
else ComprobacionesAlertasCambioEscena.mostrarAlerta("Debe haber un coche
seleccionado para obtener la información de sus multas.");
```

VALIDAR MATRÍCULA

```
//método que comprueba el formato de la matrícula
public static boolean validarMatricula(String matricula) {
   //expresión regular que verifica 4 dígitos seguidos de 3 consonantes mayúsculas
   String regex = "^[0-9]{4}[BCDFGHJKLMNPQRSTVWXYZ]{3}$";
   //comprueba si la matrícula coincide con el patrón
   return matricula != null && matricula.matches(regex);
}//validarMatricula
```

VALIDAR EMAIL

```
//método que comprueba el formato del correo electrónico
public static boolean validarEmail(String email) {
   //expresión regular que verifica un formato estándar de correo electrónico
   String regex = "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+$";
   //comprueba si el correo coincide con el patrón
   return email != null && email.matches(regex);
}//validarEmail
```

VALIDAR TELÉFONO DE 9 DÍGITOS

```
//método que comprueba el formato de un número de teléfono español de 9 cifras
public static boolean validarTelefono(String telefono) {
   //expresión regular que verifica que el número tiene exactamente 9 dígitos
   String regex = "^[0-9]{9}$";
   //comprueba si el número coincide con el patrón
   return telefono != null && telefono.matches(regex);
}//validarTelefono
```

VALIDAR IMPORTE CON DECIMALES

```
//método que valida el precio introducido (puede contener decimales, hasta 2)
public static boolean esPrecioValido(String precio) {
   return precio.matches("\\d+(\\.\\d{1,2})?");  // Acepta números con hasta 2
decimales
}//esPrecioValido
```

INICIALIZACIONES

INICIALIZAR COMBOBOX

//COMBOBOX --> le inicializo con los tipos de coche que hay
ObservableList<String> tipoCoche = FXCollections.*observableArrayList*("SUV", "Monovolumen", "Deportivo", "Pick-up", "Familiar"); //creo una lista con los tipos de coche
tipoCB.setItems(tipoCoche); //asigno la lista al ComboBox

INICIALIZAR TABLEVIEW

//TABLEVIEW --> inicializo las columnas
//inicializo las columnas del tableView (lo que hay entre "" es el getter de cada propiedad de la clase coche)
colMatricula.setCellValueFactory(new PropertyValueFactory<>("matricula"));
colMarca.setCellValueFactory(new PropertyValueFactory<>("marca"));
colModelo.setCellValueFactory(new PropertyValueFactory<>("modelo"));
colTipo.setCellValueFactory(new PropertyValueFactory<>("tipo"));
//llamo al método que muestra los coches almacenados en la base de datos y los añado a la tableView → ANTERIORMENTE HABIENDO CREADO LA CONEXIÓN A LA BD PORQUE SE TRATA DE MONGO
ArrayList<Coche> listarCoches = CocheDAO.*mostrarCoches*(); //creo un ArrayList para convertir a ObservableList
cochesOL = FXCollections.*observableArrayList*(listarCoches);
tableViewCoches.setItems(cochesOL);

INCIALIZAR LISTVIEW

//crear el ArrayList con los números que se quieren mostrar
ArrayList<Integer> numeros = new ArrayList<>(1 , 2, 3);
numeros.add(4);
numeros.add(5);
//convertir el ArrayList en un ObservableList
ObservableList<Integer> numerosOL = FXCollections.observableArrayList(numeros);
//asignar el ObservableList al ListView
numerosLV = new ListView<>(); //supuestamente ya creado como atributo del FX
numerosLV.setItems(numerosOL);

LOCALDATE EN MONGO

- Clase:
  public class LocalDateAdapter extends TypeAdapter<LocalDate> { @Override public void write(JsonWriter out, LocalDate value) throws IOException { out.value(value != null ? value.toString() : null); // Manejo de null } @Override public LocalDate read(JsonReader in) throws IOException { return LocalDate.parse(in.nextString()); }}

- CRUD:
  public boolean inertarClienteMongoDB(Clientes clientes){ Gson gson = new GsonBuilder().registerTypeAdapter(LocalDate.class, new LocalDateAdapter()).create(); if (clientes!=null){ json = gson.toJson(clientes); doc = Document.parse(json); collection.insertOne(doc); return true; } return false;}