2024

Actividad 2. Manejo de conectores

ACCESO A DATOS – UNIR DAM

TATIANA VILLA EMA

ENUNCIADO

Objetivos

Aprender a manejar JDBC mediante una pequeña aplicación de gestión de coches

Pautas de elaboración

Requerimiento 1

Se desea hacer un CRUD completo de la entidad 'Coche', pero esta vez no se trabajará con ningún fichero, se trabajará con una BBDD. Es muy importante usar el patrón DAO visto en clase. Los parámetros de conexión a la BBDD deben estar hechos en un fichero de **propiedades.**

El coche tendrá los siguientes atributos: id, marca, modelo, año de fabricación y km.

El menú mostrado será de la siguiente forma:

- Añadir nuevo coche (El ID lo incrementará automáticamente la base de datos)
- Borrar coche por ID
- Consulta coche por ID
- Modificar coche por ID (pedirá todos los valores y modificará dichos valores a partir del ID del coche)
- Listado de coches
- Terminar el programa

Valoración: 4 puntos sobre 10

Requerimiento 2

Se pide añadir la siguiente funcionalidad.

Los coches, tendrán asociados N pasajeros en él (habrá que crear la tabla pasajeros y hacer la relación pertinente). Los pasajeros tendrán los siguientes atributos, id, nombre, edad y peso. Se añadirá la opción "gestión de pasajeros" al programa principal, dicha opción nos mostrará un submenú como el que sigue

- Crear nuevo pasajero
- Borrar pasajero por id
- Consulta pasajero por id
- Listar todos los pasajeros
- Añadir pasajero a coche, el programa nos pedirá un id de un pasajero y el id de un coche, y lo añadirá al coche a nivel de base de datos. Sería una buena opción mostrar todos los coches disponibles.
- Eliminar pasajero de un coche, el programa nos pedirá un id de un pasajero y lo eliminará del coche a nivel de base de datos. Sería una buena opción mostrar todos los coches y sus pasajeros asociados.

- Listar todos los pasajeros de un coche, el programa pedirá el id de un coche, y nos mostrará todos los pasajeros asociados a él.

Valoración: 5 puntos sobre 10

Requerimiento 3

La aplicación no debe permitir que la marca y el modelo estén vacíos. Esta parte la debe de gestionar la capa gestora y seguir el modelo de tres capas visto en clase.

Valoración: 1 puntos sobre 10

Consideraciones de entrega

Para la entrega, se subirá **un documento PDF con todo lo necesario para demostrar el correcto funcionamiento de la actividad** (resultados, capturas de pantalla, ficheros, fotos, etc.). No es necesario que el documento PDF sea muy extenso, pero SÍ que incluya, al menos, la metodología de trabajo del grupo, las capturas de los resultados obtenidos con los comentarios pertinentes, y la explicación de los puntos clave de la actividad realizada. No cumplir con este punto puede llevar a suspender la actividad o a reducir considerablemente la nota final.

Además, para toda la actividad se valorará la claridad de código, la modularidad y la eficiencia de los algoritmos empleados.

Para la actividad se recomienda que los alumnos repartan las tareas a realizar, aunque también pueden proponer cada uno de ellos una solución y luego elegir cuál de ellas será la solución final mediante consenso.

Se recomienda el uso de GITHUB para realizar el trabajo y dejar el código fuente en dicha plataforma, ya que, si hay problemas con la entrega al subirla a la plataforma de EDIX, queda constancia en GITHUB de los commits hechos. Se puede subir el código fuente también comprimido a la plataforma en su lugar si así se prefiere, pero en este caso, el fichero PDF y el código comprimido (fichero .zip o .7z) deben de ir por separado.

Implementación

Arquitectura del Proyecto

Se realiza desde el principio con **el método de las tres capas**.

El enfoque de las tres capas, también conocido como arquitectura de tres capas, es una práctica común y efectiva para organizar y estructurar proyectos de software de manera modular y escalable. Consiste en dividir la aplicación en tres capas distintas: la capa de presentación (interfaz de usuario), la capa de lógica de negocio y la capa de acceso a datos. Cada capa tiene responsabilidades específicas y se comunica con las otras capas de manera controlada.

Ventajas del Enfoque de las Tres Capas:

- **Separación de Responsabilidades**: Cada capa tiene un propósito claro y se enfoca en una parte específica del sistema, lo que facilita la comprensión y el mantenimiento del código.
- **Reutilización de Código**: La separación de la lógica de negocio y el acceso a datos facilita la reutilización de componentes en diferentes partes de la aplicación.
- **Escalabilidad**: Las tres capas pueden escalar de manera independiente según sea necesario. Por ejemplo, si necesitas mejorar el rendimiento de la base de datos, puedes optimizar la capa de acceso a datos sin afectar la lógica de negocio o la capa de presentación.
- **Pruebas Unitarias**: Cada capa puede ser probada de forma independiente, lo que facilita la escritura de pruebas unitarias para garantizar la calidad del código.
- **Flexibilidad**: Si necesitas cambiar la tecnología de la capa de acceso a datos o la capa de presentación, puedes hacerlo sin afectar el resto de la aplicación, siempre que mantengas la misma interfaz entre capas.

En proyectos más grandes, donde la complejidad es mayor y la modularidad es fundamental, el enfoque de las tres capas puede ayudar a mantener el código limpio, organizado y fácil de mantener a lo largo del tiempo. Es una buena práctica que puede mejorar la eficiencia del desarrollo y la calidad del software en general.

Tecnologías Utilizadas

1. Java
2. MySQL
3. JDBC (Java Database Connectivity)**:** JDBC es una API de Java que proporciona métodos y clases para permitir que las aplicaciones Java se conecten a bases de datos y realicen operaciones CRUD (Crear, Leer, Actualizar, Eliminar). Utilizamos JDBC para interactuar con la base de datos MySQL desde nuestra aplicación Java de forma eficiente y segura.
4. Eclipse IDE
5. XAMPP
6. GitHub en https://github.com/tatvil/ADDActividad2.git

Estructura del Código

La estructura general del código de mi aplicación sigue un enfoque de arquitectura de tres capas, que separa claramente las responsabilidades y funcionalidades de la aplicación en diferentes capas. Aquí te describo cómo están organizadas las principales capas y componentes de mi aplicación:

1. **Capa de Presentación (Interfaz de Usuario)**:
    - Esta capa se encarga de interactuar con el usuario y presentar la información de manera visual.
    - Incluye clases y componentes relacionados con la interfaz de usuario, como formularios, menús, etc.
    - Aunque me he visto tentada a utilizar tecnologías como Java Swing, JavaFX, para la creación de la interfaz de usuario, he optado por realizar los menús en consola, porque el proyecto no es muy grande y así lo permitía. Además, lo que queríamos practicar era el acceso a datos.
    - El paquete se llama “vista”.
2. **Capa de Lógica de Negocio**:
    - Esta capa contiene la lógica y las reglas de negocio de la aplicación.
    - Aquí se encuentran las clases y métodos que procesan la información, realizan cálculos y aplican reglas específicas del dominio del problema.
    - Es independiente de la capa de presentación y la capa de acceso a datos, lo que facilita la reutilización y mantenimiento del código.
    - El paquete se llama modelo.negocio
3. **Capa de Acceso a Datos**:
    - Esta capa se encarga de interactuar con la base de datos y gestionar las operaciones de lectura, escritura, actualización y eliminación de datos.
    - Incluye clases y métodos que utilizan JDBC para establecer conexiones con la base de datos y ejecutar consultas SQL.
    - Separar esta capa del resto de la aplicación ayuda a garantizar la modularidad y la flexibilidad del sistema, permitiendo cambios en la tecnología de almacenamiento de datos sin afectar otras partes de la aplicación.
    - El paquete se llama modelo.persistencia
4. Configuración de los datos de la base de datos en un fichero

5. **Paquetes, Clases y Organización**:

    Los componentes de cada capa se organizan en paquetes o módulos lógicos según su funcionalidad y relación.

    La estructura general del código sigue el principio de separación de preocupaciones y modularidad, lo que facilita el desarrollo, mantenimiento y escalabilidad de la aplicación. Además, esta arquitectura permite realizar pruebas unitarias de manera más efectiva y proporciona una base sólida para futuras expansiones y mejoras del sistema.

Los paquetes son los siguientes:

1. crearbbdd: Clases que crean la base de datos y las tablas que se utilizan en el programa.
    1. CrearMySQL: Esta clase se encarga de crear la base de datos y las tablas necesarias si no existen. Utiliza JDBC para establecer una conexión con la base de datos MySQL y ejecutar las consultas SQL necesarias para crear la base de datos y las tablas.
2. modelo.entidad: Clases con los objetos que se utilizarán para recibir los datos de la base de datos.
    1. Coche.java: Representa la entidad de un coche en la aplicación. Contiene atributos y métodos para acceder y manipular los datos de un coche.
    2. Pasajero.java: Propósito: Representa la entidad de un pasajero en la aplicación. Contiene atributos y métodos para acceder y manipular los datos de un pasajero.
3. modelo.negocio: Gestión de las reglas del negocio. Es decir, gestiona o verifica las reglas que tienen que cumplir los datos de las tablas o los objetos.
    1. GestorCoche.java: Clase que gestiona los coches y sus reglas de negocio. Estas reglas se aplicarán antes de llamar a la BBDD.
    2. GestorPasajero.java: Clase que gestiona los pasajeros y sus reglas de negocio. Estas reglas se aplicarán antes de llamar a la BBDD
4. modelo.persistencia: Capa de acceso a datos de una aplicación. En este contexto, la capa de persistencia se encarga de interactuar con la base de datos o cualquier otro mecanismo de almacenamiento permanente para realizar operaciones de lectura, escritura, actualización y eliminación de datos.
    1. DaoCocheMySQL.java: Clase que gestiona la persistencia de los coches en la base de datos MySQL. La clase implementa la interfaz DaoCoche. La persistencia de los coches se hace en una tabla llamada coches que tiene los siguientes campos: id, marca, modelo, fabricación, km. La tabla coches tendrá un campo id que será la clave primaria y se generará de manera automática.
    2. DaoPasajerosMySQL.java: Clase que gestiona la persistencia de los pasajeros en la base de datos MySQL. La clase implementa la interfaz DaoPasajero. La persistencia de los pasajeros se hará en una tabla llamada pasajeros. La tabla pasajeros tendrá los siguientes campos: id, nombre, edad, peso, id_coche. El id_coche es una clave foránea que hace referencia al id de la tabla coches. La tabla pasajeros tendrá un campo id que será la clave primaria y se generará de manera automática
    3.

5. modelo.persistencia.interfaces: Las interfaces que definen los contratos para las operaciones de persistencia que deben implementarse en las clases concretas de acceso a datos.
    1. DaoCoche.java: Interfaz que define los métodos que se van a poder realizar sobre la BBDD para la entidad coche
    2. DaoPasajero.java: Interfaz que define los métodos que se van a poder realizar sobre la BBDD para la entidad pasajero
6. vista: La capa de presentación de una aplicación. Es la responsable de la interfaz de usuario y cómo se presenta la información al usuario final.
    1. MainTresCapas.java: Clase que gestiona la vista de la aplicación mediante el uso de la consola de comandos de Java (System.out) y la entrada de datos (System.in). Presenta la dinámica de la interacción con el usuario y recoge sus respuestas, según las opciones que se le presenten en el menú principal y el menú de pasajeros realizara las acciones oportunas. Se comunica con la capa de negocio y la capa de persistencia.
7. config.properties: Contiene los datos de acceso a la base de datos.

Capturas de Pantalla de cómo funciona la aplicación

```
Bienvenidos a nuestra CRUD de coches
Elija una opcion:
1. Añadir nuevo coche
2. Borrar coche por ID
3. Consulta coche por ID
4. Modificar coche por ID
5. Listado de coches
6. Asignar pasajero a coche
0. Terminar el programa
```

```
Introduzca los datos del coche (marca, modelo, fabricacion y km)
Ferrari
Testarrosa
1987
105000
Coche dado de alta
```

```
2
2. BORRAR COCHE
Introduzca los datos del coche a borrar (id)
8
```

```
3. CONSULTAR COCHE
Introduzca los datos del coche a consultar (id)
3
coche [id=3, marca=Seat, modelo=Ibiza, fabricacion=2016, km=153321]
```

```
4. MODIFICAR COCHE
Introduzca los datos del coche a modificar (id, marca, modelo, fabricacion y km)
3
Seat
Ibiza
2007
153231
Coche modificado
```

```
5. LISTADO DE COCHES
[coche [id=2, marca=Volkswagen, modelo=Passat, fabricacion=2022, km=54989]
, coche [id=3, marca=Seat, modelo=Ibiza, fabricacion=2016, km=153321]
, coche [id=4, marca=Nissan, modelo=Quasquais, fabricacion=2007, km=98023]
, coche [id=5, marca=Nissan, modelo=Juke, fabricacion=2007, km=87891]
, coche [id=7, marca=BMW, modelo=FullHD, fabricacion=2007, km=2]
, coche [id=8, marca=Ferrari, modelo=Testarrosa, fabricacion=1987, km=105000]
]
```

```
6
Elija una opcion:
1. Crear nuevo pasajero
2. Borrar pasajero por id
3. Consulta pasajero por id
4. Listar todos los pasajeros
5. Añadir pasajero a coche
6. Eliminar pasajero de un coche
7. Listar todos los pasajeros de un coche
0. Atras (menu coches)
```

```
1. CREAR NUEVO PASAJERO
Introduzca los datos del pasajero (nombre, edad, peso)
Felix
35
70
Alta de pasajero [id=0, nombre=Felix, edad=35, peso=70, coche=null]
```

```
2. BORRAR PASAJERO
Introduzca los datos del pasajero a borrar (id)
17
Baja de 17
```

```
Introduzca los datos del pasajero a consultar (id)
17
pasajero [id=17, nombre=Felix, edad=35, peso=70, coche=coche [id=0, marca=null, modelo=null, fabricacion=0, km=0]
]
```

```
4. LISTAR PASAJEROS
Listado de pasajeros
[pasajero [id=1, nombre=Tatiana, edad=52, peso=76, coche=coche [id=3, marca=Seat, modelo=Ibiza, fabricacion=2007, km=153231]
]
, pasajero [id=3, nombre=Juan, edad=53, peso=80, coche=coche [id=2, marca=Volkswagen, modelo=Passat, fabricacion=2022, km=54989]
]
, pasajero [id=6, nombre=Andrea, edad=23, peso=80, coche=coche [id=3, marca=Seat, modelo=Ibiza, fabricacion=2007, km=153231]
]
, pasajero [id=8, nombre=Nico, edad=19, peso=50, coche=coche [id=3, marca=Seat, modelo=Ibiza, fabricacion=2007, km=153231]
]
, pasajero [id=11, nombre=Leo, edad=19, peso=50, coche=coche [id=3, marca=Seat, modelo=Ibiza, fabricacion=2007, km=153231]
]
, pasajero [id=12, nombre=Arantxa, edad=49, peso=45, coche=coche [id=4, marca=Nissan, modelo=Quasquais, fabricacion=2007, km=98023]
]
, pasajero [id=13, nombre=Luz, edad=20, peso=40, coche=coche [id=4, marca=Nissan, modelo=Quasquais, fabricacion=2007, km=98023]
]
, pasajero [id=14, nombre=Jose, edad=83, peso=75, coche=coche [id=5, marca=Nissan, modelo=Juke, fabricacion=2007, km=87891]
]
, pasajero [id=15, nombre=Mercedes, edad=80, peso=40, coche=coche [id=5, marca=Nissan, modelo=Juke, fabricacion=2007, km=87891]
]
, pasajero [id=16, nombre=Jose, edad=83, peso=78, coche=coche [id=0, marca=null, modelo=null, fabricacion=0, km=0]
]
, pasajero [id=17, nombre=Felix, edad=35, peso=70, coche=coche [id=0, marca=null, modelo=null, fabricacion=0, km=0]
]
]
```

```
5. AÑADIR PASAJERO A COCHE
Introduzca el id del pasajero y el id del coche
17
9
Modificado pasajero [id=17, nombre=Felix, edad=35, peso=70, coche=coche [id=9, marca=Ford, modelo=Fiesta, fabricacion=2023, km=100]
]
```

```
Introduzca el id del coche
9
Listado de pasajeros
[pasajero [id=1, nombre=Tatiana, edad=52, peso=76, coche=coche [id=9, marca=Ford, modelo=Fiesta, fabricacion=2023, km=100]
]
, pasajero [id=17, nombre=Felix, edad=35, peso=70, coche=coche [id=9, marca=Ford, modelo=Fiesta, fabricacion=2023, km=100]
]
]
```

Problemas y Soluciones

Me ha resultado más fácil que otras actividades. A lo mejor porque ya había realizado accesos a bases de datos con otros lenguajes de programación como PHP. Por otra parte me apasiona el mundo de los datos, el IoT y la IA, y es algo en lo que he hecho hincapié en mi vida, por ejemplo durante la pandemia he estado monitorizando la incidencia del COVID en distintos municipios de España, para uso personal, y así decidir donde y cuando aislarme, pues soy persona de riesgo.

Nunca antes había estructurado una aplicación con modelo de tres capas, pero siguiendo los ejemplos de WorkSpaceJava ha sido muy fácil.

Para probar el programa

Para probar el funcionamiento del programa, puedes descargarlo desde el siguiente enlace: ADDActividad2.

Antes de utilizarlo, es necesario inicializar la base de datos. Para ello, debes activar XAMPP y ejecutar el programa "CrearMySQL.java", ubicado dentro del paquete "crearBBDD".

Los datos de configuración de la base de datos se encuentran en el archivo "config.properties".

Para ejecutar la actividad, simplemente ejecuta el archivo "MainTresCapas.java" ubicado en la carpeta "vista". Este archivo abrirá el menú con todas las opciones descritas en los requerimientos 1 y 2.