EJERCICIO 1

Para este ejercicio usaremos la tabla PERSONAS:

```
-- MySQL
CREATE TABLE IF NOT EXISTS personas (
  DNI varchar(50) NOT NULL,
  NOMBRE varchar(50) NOT NULL,
  APELLIDOS varchar(100) NOT NULL,
  FECHA_NACIMIENTO date DEFAULT NULL,
  PRIMARY KEY (DNI)
);
```

```
-- ORACLE
CREATE TABLE personas (
  DNI varchar2(50) NOT NULL,
  NOMBRE varchar2(50) NOT NULL,
  APELLIDOS varchar2(100) NOT NULL,
  FECHA_NACIMIENTO DATE DEFAULT NULL,
  PRIMARY KEY (DNI)
);
```

Ejecuta el script en tu BBDD para crear la tabla. Inserta una o dos personas manualmente como prefieras (con un script, con SQLDeveloper, HeidiSQL, etc.)

A continuación:

Realiza un programa que pida al usuario un DNI, consulte en la BBDD la persona con ese DNI y muestre sus datos por pantalla.

Realiza el programa organizado en las siguientes clases:

- Crea una clase Persona
- Crea una clase PersonasService con el método consultarPersona()
- Crea una clase App que se encargue de solicitar al usuario el DNI y llame al método consultarPersona() cuando lo requiera.

Si hay algún error al conectar con la BBDD o ejecutar la consulta, tendrás que mostrárselo al usuario con un texto amigable.

EJERCICIO 2

Completa el ejercicio anterior solicitando después al usuario un filtro para buscar personas. Consulta en la BBDD todas las personas cuyo nombre o apellidos contengan este filtro. Luego muestra al usuario el listado de los resultados encontrados.

Tendrás que crear un método en PersonasService llamado buscarPersonas() que reciba el filtro y devuelva una lista de personas.

Trata los errores del mismo modo.

Intenta dividir tu clase App en método para que no esté todo “apiñado” en el main

EJERCICIO 3

Amplía el programa anterior para que, después de la última parte, solicite al usuario los datos de una persona y lo inserte en BBDD. Tendrás que crear en PersonasService un método llamado insertarPersona() que reciba una Persona y la inserte. Trata los errores con la BBDD del mismo modo.

Antes de intentar insertar, debes asegurar que todos los datos están completos. Para ello, crea un método "validar()" en la clase Persona que lance una excepción si no es así. Por ejemplo, DatosIncompletosException. Si salta la excepción, contrólala y vuelve a pedir los datos al usuario.

EJERCICIO 4

Amplía el programa anterior para que, nada más arrancar, aparezca un menú con las 3 opciones disponibles más una cuarta opción para salir del programa. Cada vez que el usuario escoja una de las opciones del menú, se ejecutará esa opción, y luego se le volverá a mostrar el menú.

EJERCICIO 5

Amplía el programa anterior incluyendo en el menú dos opciones más:

- Actualizar persona → Debe pedir los datos de una persona y actualizarla en BBDD.
- Borrar persona → Debe pedir el dni de una persona y borrarla de BBDD.

Utilizar un PreparedStatement en lugar de un Statement para que sea más fácil escribir la query.

EJERCICIO 6

Amplia el programa anterior para que, si el usuario indica un DNI que no existe a la hora de actualizar o borrar una persona, se le indique por pantalla que el DNI indicado no pertenece a ningún registro de la base de datos.

Para evitar hacer una consulta adicional, utiliza el método executeUpdate().

EJERCICIO 7

Crea en PersonasService un método que permita insertar una lista de Personas en la BBDD. Se llamará insertarPersonas() y recibirá una lista. Debe llamar al método que ya tenemos creado previamente (insertarPersona) N veces (según el número de personas que haya en la lista).

Importante: si hay algún error al guardar alguna persona, la transacción debe deshacerse por completo. Es decir, si hay algún error, no se debe insertar ninguna persona (tendrás que hacer un rollback)

¿Cómo hacemos esto?

- Tendrás que dividir el método insertarPersona() en dos métodos.
  - Uno privado que recibe una conexión por parámetro y la usa para todo lo demás (no la crea ni la cierra)
  - Otro que no recibe nada, público, crea una conexión, llama al anterior, y cierra la conexión.
- Cuando hayas hecho eso, tu nuevo método insertarPersonas() podrá crear una conexión y, con un bucle, ir llamando al insertarPersona privado para insertarlas.

Pruébalo creando una clase Test que cree 3 personas y llame al servicio. Prueba también a hacer que la segunda o tercera persona tenga un nombre muy largo para que provoque un error en la base de datos y verificar que no se inserta ninguna persona.

EJERCICIO 8

Crea otro método en PersonasService llamado borrarPersonasA() que borre de la base de datos todas las personas mayores de edad.

El método tendrá que implementarse del siguiente modo:

1. Consultar todas las personas de la BBDD (utiliza el método que ya existe)
2. Recorrer todas las personas y comprobar si son mayores de edad o no (puedes crear un método en la clase Persona que te lo indique)
3. Si la persona es mayor de edad, bórrala. Utiliza el método que ya existe, pero tendrás que dividirlo de nuevo para compartir la conexión.
4. El servicio tendrá que devolver la cantidad de personas borradas.

Pruébalo añadiendo a tu programa una opción más de menú que sea “Borrar adultos” y que muestre al usuario la cantidad de personas borradas al terminar.

Importante: la operación debe hacerse en una única transacción. Es decir, si hay algún error en cualquier momento, no se borrará nada.

EJERCICIO 9

Crea otro método en PersonasService llamado borrarPersonasB() que haga lo anterior pero en una única instrucción DELETE. Para ello, calcula con LocalDate la fecha límite de nacimiento para saber si las personas son o no mayores de edad y úsala como filtro en el SQL.

Prueba el método igual que antes.

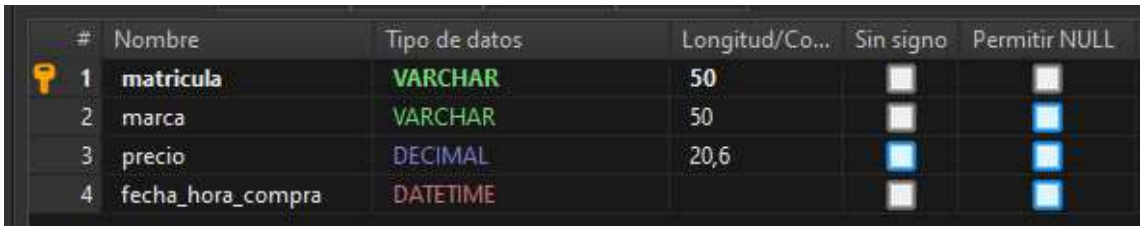
EJERCICIO 10 (opcional - repaso)

Para este ejercicio usaremos la tabla COCHES:

```
-- MySQL
CREATE TABLE IF NOT EXISTS coches (
  matricula varchar(50) NOT NULL,
  marca varchar(50) NOT NULL,
  precio decimal(20,6) NOT NULL,
  fecha_hora_compra datetime NOT NULL,
  PRIMARY KEY (matricula)
);
```

```
-- ORACLE
CREATE TABLE coches (
  matricula varchar2(50) NOT NULL,
  marca varchar2(50) NOT NULL,
  precio number(20,6) NOT NULL,
  fecha_hora_compra DATE NOT NULL,
  PRIMARY KEY (matricula)
);
```

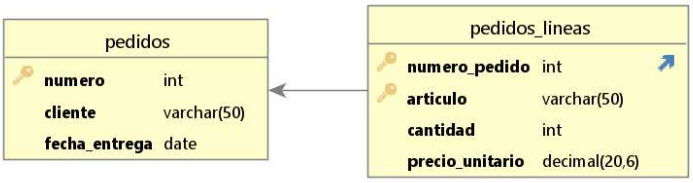
Haz un CRUD similar al realizado en los ejercicios anteriores (del ejercicio 1 al 6) que permita insertar/actualizar/borrar/consultar/buscar datos de la tabla COCHES. La consulta se podrá realizar únicamente por matrícula y la búsqueda por marca



#	Nombre	Tipo de datos	Longitud/Co...	Sin signo	Permitir NULL
1	matricula	VARCHAR	50	☐	☐
2	marca	VARCHAR	50	☐	☑
3	precio	DECIMAL	20,6	☑	☑
4	fecha_hora_compra	DATETIME		☐	☑

EJERCICIO 11

Realiza un nuevo programa que nos permita gestionar los pedidos que se realizan en un supermercado. El programa trabajará con estas dos tablas:




```
-- MySQL
CREATE TABLE IF NOT EXISTS pedidos (
  numero int(11) NOT NULL,
  cliente varchar(50) NOT NULL,
  fecha_entrega date NOT NULL,
  PRIMARY KEY (numero)
);
CREATE TABLE IF NOT EXISTS pedidos_lineas (
  numero_pedido int(11) NOT NULL,
  articulo varchar(50) NOT NULL,
  cantidad int(11) NOT NULL,
  precio_unitario decimal(20,6) NOT NULL,
  PRIMARY KEY (numero_pedido,articulo),
  CONSTRAINT FK_pedido FOREIGN KEY (numero_pedido) REFERENCES pedidos (numero)
ON DELETE NO ACTION ON UPDATE NO ACTION
);
```

```
-- ORACLE
CREATE TABLE pedidos (
  numero NUMERIC(6,0) NOT NULL,
  cliente varchar2(50) NOT NULL,
  fecha_entrega date NOT NULL,
  CONSTRAINT PK_PEDIDOS PRIMARY KEY (numero)
);
CREATE TABLE pedidos_lineas (
  numero_pedido NUMERIC(6,0) NOT NULL,
  articulo varchar2(50) NOT NULL,
  cantidad NUMERIC(6,0) NOT NULL,
  precio_unitario NUMERIC(10,6) NOT NULL,
  CONSTRAINT PK_PEDIDOS_LINEAS PRIMARY KEY (numero_pedido, articulo),
  CONSTRAINT FK_pedido FOREIGN KEY (numero_pedido) REFERENCES pedidos(numero)
);
```

Al arrancar, el programa mostrará un menú con tres opciones:

a) Crear nuevo pedido: Deberá solicitar los datos del pedido y cuántas líneas tendrá. Luego pedirá también todas esas líneas de pedido. Al terminar, tendrá que guardar todo en BBDD y volver a mostrar el menú. Si hay algún error, se mostrará por la interfaz de usuario. Importante: la transacción debe ser única para asegurar la consistencia de los datos.

b) Ver pedido: Deberá solicitar el número de pedido que se quiere consultar y mostrarlo por pantalla. Si hay algún error, se mostrará por la interfaz.

c) Salir: Saldrá de la aplicación.

Divide tus clases del siguiente modo:

- Modelo:
  - Clase PedidoLinea → Que tenga los datos de la tabla PEDIDOS_LINEAS
  - Clase Pedido → Que tenga todos los datos de la tabla PEDIDOS y una lista de PedidoLinea (las líneas del pedido)

- Clase PedidosService con estos métodos:
  - consultarLineasPedido() → consulta todas las líneas de pedido (tabla PEDIDOS_LINEAS) filtrando por NUMERO_PEDIDO. Devolverá una lista.
  - consultarPedidoCompleto() → consulta los datos de la tabla Pedido filtrando por número pedido. Luego consulta sus líneas usando el método anterior y así obtiene un pedido completo y lo devuelve. Si el pedido no existe, lanzará una nueva excepción llamada NotFoundException. Si hay cualquier error con la BBDD, lanzará PedidoException.
  - insertarLineaPedido() → Recibe una conexión y una línea de pedido. Inserta los datos de la línea en la tabla PEDIDOS_LINEA.
  - crearPedidoCompleto() → recibe un objeto Pedido. Y tiene que:
    - Insertar los datos del pedido en la tabla PEDIDOS.
    - Completar los datos que falten en las líneas de pedido
    - Insertar todas las líneas de pedido usando el método anterior
    - Si hay algún error con la BBDD, lanzará PedidoException.
- App e interfaz:
  - Crea una clase App con un main para arrancar el programa. Tendrá que "hablar" con el usuario y utilizar las clases anteriores cuando sea necesario. Puede ser recomendable crear también una clase Menu o Pantalla para simplificar la clase App

NOTA: OJO con el control de transacciones en BBDD (commit, rollback…). Los errores al cerrar conexión o statement pueden ser ignorados.

EJERCICIO 13

Modifica tu programa para que al mostrar un pedido muestre también el subtotal, el IVA y el total, todos redondeados a 2 decimales con HALF_UP. La forma de calcularlos sería la siguiente:

- Subtotal = Sumatorio de todas las líneas de cantidad x precio unitario
- IVA = 21% del Subtotal
- Total = Suma del subtotal y el IVA

Para ello:

- crea estos tres atributos en la clase Pedido con sus get/set
- crea un método en la clase Pedido llamado "calcularTotales()" que haga el cálculo
- cuando consultes un Pedido de BBDD, llama al método anterior para calcular sus totales

EJERCICIO 14

Modifica tu programa para añadir una opción adicional que sea "Cambiar fecha de entrega". Se deberá solicitar al usuario el número de pedido y la nueva fecha de entrega. Habrá un método en PedidosService que reciba esos dos datos y se encargue de modificar en BBDD la fecha de entrega. Si el número de pedido indicado no existe, se lanzará un NotFoundException desde el servicio y se informará al usuario por interfaz.

EJERCICIO 15

Modifica tu programa para añadir una opción adicional que sea "Borrar pedido". Se deberá solicitar al usuario el número de pedido que quiere borrar.

Habrá un método en PedidosService que, recibiendo ese dato únicamente, borre los datos asociados al pedido en las dos tablas.

Si no hay ningún pedido con el número indicado, se lanzará NotFoundException desde el servicio y se informará al usuario por interfaz.

EJERCICIO 16 (avanzado)

Modifica tu programa para añadir una opción adicional que sea "Buscar pedidos". Al elegir esta opción tendrá que aparecer otro submenú con tres opciones:

a) Buscar por cliente
b) Buscar por fecha
c) Volver al menú principal

La opción "Volver al menú principal" mostrará de nuevo el menú inicial.

La opción de buscar por cliente solicitará un texto por el que filtrar en BBDD para mostrar al usuario todos los pedidos que tengan un cliente que contenga dicho texto. Sólo será necesario mostrar un listado indicando número de pedido, cliente y fecha de todos los pedidos encontrados.

La opción de buscar por fecha solicitará una fecha desde y otra fecha hasta. Filtraremos en BBDD para mostrar al usuario todos los pedidos cuya fecha de entrega esté comprendida entre ese intervalo (ambos extremos inclusive). Al igual que en el caso anterior, bastará con mostrar un listado indicando número de pedido, cliente y fecha de todos los pedidos encontrados.