Ejercicio 1: Dada una tabla clientes con id, nombre y apellidos crear un servicio con los siguientes métodos y probarlos desde el main.

- getClientes: obtiene la lista de todos los clientes
- getCliente: dado un id, obtiene sus datos de clientes.
- insertaCliente: inserta los datos de un cliente.
- deleteCliente: dado un id de un cliente, lo elimina.
- actualizarCliente: recibe un cliente y lo actualiza
- getClientesNombre: dado un nombre, obtiene la lista de todos los clientes cuyo nombre contenga el nombre a buscar sin tener en cuenta mayúsculas o minúsculas.
- getClienteNombreyApellidos: dado un nombre y apellidos, obtiene la lista de clientes que sean exactamente igual (ambos).
- getClientesOrdenadosAPellidos: devuelve todos los clientes ordenados por apellido de forma ascendente.
- getNumeroCliente: recibe un nombre de un cliente y devuelve todos los que coincide exactamente con ese nombre.
- getClientesPrefijoSufijo: recibe dos cadenas cad1,cad2, y devuelve los clientes cuyo nombre empieza por cad1 y sus apellidos terminan por cad2.

Script:

```
CREATE TABLE `clientes` (
  `id` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `nombre` VARCHAR(20) NULL,
  `apellidos` VARCHAR(45) NULL
);
```

NOTA: El campo id es autonumérico.

Crear los paquetes modelos, servicio y repositorio de esta forma:

```
com.ejercicios.ejercicio1
  ├── modelos
  |    └── Cliente.java
  ├── repositorio
  |    └── ClienteRepository.java
  ├── servicio
  |    ├── ClienteService.java
  |    └── ClienteServiceImpl.java
  └── Ejercicio1Application.java
```

Ejercicio 2: Dada la tabla product,

```
CREATE TABLE `product` (
  `id` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `name` VARCHAR(45) not NULL,
  `price` FLOAT NOT NULL
);
```

Crear un servicio con los siguientes métodos.

- Encontrar productos por nombre y precio: Devolver todos lo productos cuyo nombre empiece por una cadena y su precio sea inferior a una cantidad.
- Dado un nombre devuelve la lista de productos donde aparezca contenido dicho nombre. (ignorar mayúsculas).
- Encontrar productos por rango de precios: Dado dos precios, obtener la lista de productos cuyo precio se encuentre en dicho rango.
- Insertar una lista de productos: Dada una lista de productos insertarlos en base de datos. Asegurándote de que los productos nuevos se inserten y los productos existentes se actualicen.

Probar lo siguiente en el main:

- Crear una lista de 6 productos
- Guardar la lista en la base de datos
- Obtener y escribir los productos (todos los datos), que comienzan por L y su precio es inferior a 5 euros
- Obtener y escribir los productos que contienen en el nombre 'li'
- Obtener y escribir los productos cuyo precio está entre 1 y 3,5 euros
- Insertar una lista de 3 productos, donde 2 no esté y otro sí.
- Encontrar y mostrar los productos cuyo precio esté entre 0 y 100 euros.

Ejercicio 3:

Desarrollar un servicio para gestionar los vehículos de una flota. Permitirá realizar un CRUD completo (Crear, Leer, Actualizar y Eliminar) para gestionar la información de los vehículos. Además, deberá implementar métodos adicionales para realizar operaciones específicas relacionadas con el estado y el uso de los vehículos.

Script:

```
CREATE TABLE vehiculo (
   id BIGINT AUTO_INCREMENT PRIMARY KEY,
   marca VARCHAR(100) NOT NULL,
   modelo VARCHAR(100) NOT NULL,
   anyo INT NOT NULL,
   matricula VARCHAR(50) NOT NULL UNIQUE,
   estado VARCHAR(50) NOT NULL,
   kilometraje DOUBLE NOT NULL
);
```

Requerimientos de CRUD:

- Crear Vehículo: Permitir la creación de un nuevo vehículo con los atributos especificados. Debe devolver el vehículo insertado.
- Consultar Vehículo: Devuelve el vehículo específico mediante su id

- Consultar la lista de vehículos: Devuelve la lista de todos los vehículos.
- Actualizar Vehículo: recibe un vehículo a actualizar y un id del vehiculo
- Eliminar Vehículo: Permitir la eliminación de un vehículo mediante su id.

Métodos Adicionales:

- Actualizar Estado de un vehículo: Recibe el nuevo estado de un vehículo y su id, y cambia el estado de un vehículo (por ejemplo, cambiar de "Disponible" a "En Mantenimiento").
- Registrar Kilometraje: Recibe los kms nuevos del vehiculo y su id.
- Consultar Vehículos por Estado: Recibe un estado, y obtiene la lista de todos los vehículos que se encuentren en un estado específico (por ejemplo, todos los "En Uso").
- Filtrar Vehículos por Año: Devuelve una lista de vehículos fabricados dentro de un rango de dos años. Necesitará un año inicial y otro final para la consulta.
- masKms: Devuelve el vehículo con más kilómetros

Probar lo siguiente en el main:

- Crear dos vehículos e insertarlos en BD
- Mostrar los datos de todos los vehículos
- Mostrar los vehículos que están en estado “En Mantenimiento”
- Mostrar el vehículo con más kilómetros.

Ejercicio 4: Vamos a gestionar hoteles.

Script:

```
CREATE TABLE hotel (
   id BIGINT AUTO_INCREMENT PRIMARY KEY,
   nombre VARCHAR(255) NOT NULL UNIQUE,
   direccion VARCHAR(255) NOT NULL,
   estrellas INT NOT NULL CHECK (estrellas BETWEEN 1 AND 5),
   telefono VARCHAR(50) NOT NULL,
   pagina_web VARCHAR(255)
);
```

Crear un servicio con los siguientes métodos:

- buscarPorEstrellas: recibe un numero de estrellas y devuelve la lista de los hoteles con ese número de estrellas.
- buscarRangoEstrellas: Obtiene una lista de hoteles cuyo número de estrellas se encuentre dentro de un rango específico mínimo y máximo.
- TresMasEstrellas: devuelve la lista con los 3 hoteles con mas estrellas.
- buscarNombreoDireccion: Devuelve la lista de hoteles buscando una cadena por nombre o dirección
- contarPorEstrellas: devuelve el numero de hoteles de un determinado numero de estrellas que se pasa por parámetro.
- HotelesSinWeb: devuelve la lista de hoteles que no tienen página web registrada.

- InsertarHotel: recibe un hotal y lo inserta en BD

- obtenerHotelesEstrellasDescyNombreAsc: devuelve la lista de todos los hoteles que vengan ordenados por estrellas descendentes y nombre ascendente.
  Hacerlo con consulta derivada y utilizando JPQL.

Crear un main:

- Crear 5 hoteles e insertarlos en BD
- Mostrar el listado de hoteles
- Mostrar los hoteles de 5 estrellas
- Mostrar los hoteles con mas de tres estrellas
- Mostrar los hoteles que contienen 'Luna' en el nombre o en la dirección
- Devolver el número de hoteles con 5 estrellas
- Mostrar la lista de hoteles sin página web.

Ejercicio 5:

Vamos a gestionar los animales de un zoológico. Crear un servicio con los métodos de un CRUD completo.

Script:

```
CREATE TABLE animales (
id BIGINT AUTO_INCREMENT PRIMARY KEY,
nombre VARCHAR(255) NOT NULL UNIQUE,
especie VARCHAR(255) NOT NULL,
edad INT NOT NULL,
habitat VARCHAR(255) NOT NULL,
fechaIngreso DATE NOT NULL
);
```

Además el servicio debe tener los siguientes métodos adicionales:

- animalesPorEspecie: Dada una especie, devolver la lista de animales
- animalesPorEdad: Dada una edad, devolver la lista de animales que tienen esa edad. Solo interesa saber la especie, el nombre y su habitat.
- animalesDesdeAnyo: devuelve los animales que han entrado en el zoo en los últimos x años. Recibe el año como parametro. Solo interesa saber la fecha, la especie y el nombre.
- animalesAntesDeFecha(LocalDate fecha): Animales que ingresaron antes de una fecha dada.
- animalesDespuesDeFecha(LocalDate fecha): Animales que ingresaron después de una fecha dada.
- animalesEntreFechas(Date inicio, Date fin): Animales ingresados entre dos fechas.
- AnimalesOrdenadosPorFechaIngreso(): Lista todos los animales ordenados por fecha de ingreso ascendente o descendente. (debe recibirlo como parámetro)
- contarAnimalesIngresadosEnAnyo(int año): Devuelve Número de animales ingresados en un año específico.

Crear en el main:

- Crear 4 animales e insertarlos en BD.
- Mostrar los animales de la especie 'León'
- Mostrar los animales de 7 años
- Mostrar los animales que ingresaron en los últimos 4 años
- Mostrar los animales anteriores al 1 de enero de 2020
- Mostrar los animales posteriores a 1 de enero de 1019
- Mostrar los animales ordenados por fecha de ingreso ascendente.
- Mostrar el número de animales del año 2019.

Ejercicio 6: Dadas las tablas Cliente y Direccion:

CREATE TABLE cliente (id INT NOT NULL AUTO_INCREMENT, nombre VARCHAR(255) NOT NULL, direccion_id INT, PRIMARY KEY (id), FOREIGN KEY (direccion_id) REFERENCES direccion(id));	CREATE TABLE direccion (id INT NOT NULL AUTO_INCREMENT, calle VARCHAR(255) NOT NULL, ciudad VARCHAR(255) NOT NULL, PRIMARY KEY (id));

Desarrollar una aplicación en Spring Boot que gestione una relación One-to-One unidireccional Cliente a Direccion (Un cliente conoce su dirección, pero a partir de la dirección no sabemos quién es el cliente) Utilizar JPA para la persistencia de datos. Implementar el servicio y repositorio de manera independiente. El servicio debe tener los siguientes métodos:

1. Obtener la lista de todos los clientes.
2. Obtener los datos de un cliente dado su ID.
3. Insertar un nuevo cliente.
4. Actualizar los datos de un cliente: recibe el id y el cliente actualizar.
5. Eliminar un cliente dado su ID. Debe devolver true o false.
6. Modificar la dirección de un cliente. Recibe el id del cliente y la nueva dirección (Direccion).
7. Actualizar la ciudad a 'Sevilla' de todos los clientes cuyo nombre empiece por 'A' o 'a'
8. Crear un nuevo método similar al anterior, pero para que la ciudad y la letra de inicio sean parámetros. Es decir, sirva para cualquier letra y ciudad.
9. Buscar Clientes por Ciudad. Dada una ciudad, devolver el listado de clientes de dicha ciudad.

Realizar las siguientes pruebas:

1. Insertar el cliente con nombre Pepe Ruiz y dirección calle Sol de Oviedo.
2. Insertar el cliente con nombre Andrés Ramírez y dirección calle Mar de Cádiz.
3. Mostrar la lista de todos los clientes
4. Obtener los datos del cliente con el id de Pepe Ruiz.
5. Dado el id de Pepe Ruiz, actualizar sus datos: Pepe José Ruiz, calle luna de Madrid.
6. Mostrar la lista de todos los clientes y comprobar la actualización.
7. Volver a añadir a otro cliente con los mismos datos de Andrés Ramírez.

8. Modificar la dirección del primer Andrés: Calle Estrella de Málaga, usando el método 6.
9. Actualizar a Sevilla todos los clientes cuyo nombre empiece por 'A' o 'a'. Comprobar
10. Actualizar a Granada todos los clientes cuyo nombre empiece por P. Comprobar
11. Mostrar la lista de todos los clientes de Sevilla.
12. Eliminar al último Andrés insertado.
13. Mostrar de nuevo la lista de clientes.

Ejercicio 7: Repetir el ejercicio anterior teniendo en cuenta la relación one to one bidireccional. ¿Qué cambios hay que hacer respecto el ejercicio anterior?

Ejercicio 8: Desarrollar una aplicación en Spring Boot que gestione una relación One-to-One unidireccional entre las entidades Perfil y Usuario. Utilizar Hibernate para la persistencia de datos. Crear para las clases del modelo, constructores vacío y con todos los parámetros así como el método toString().

CREATE TABLE usuario (id INT NOT NULL AUTO_INCREMENT, nombre VARCHAR(255) NOT NULL, email VARCHAR(255) NOT NULL, perfil_id INT UNIQUE, PRIMARY KEY (id), FOREIGN KEY (perfil_id) REFERENCES perfil(id));	CREATE TABLE perfil (id INT NOT NULL AUTO_INCREMENT, bio TEXT NOT NULL, estado VARCHAR(30), PRIMARY KEY (id));

El servicio debe tener los siguientes métodos:

1. Obtener la lista de todos los usuarios.
2. Obtener los datos de un usuario dado su ID.
3. Insertar un nuevo usuario: recibe un usuario.
4. Actualizar los datos de un usuario: recibe un usuario
5. Eliminar un usuario dado su ID.
6. Actualizar el perfil de un usuario. Recibe el id del usuario y su nuevo perfil. Devuelve el usuario actualizado.
7. Obtener el perfil de un usuario dado su ID.
8. Actualizar el estado de un perfil a 'DISPONIBLE' dado el id del usuario.
9. Obtener la lista de usuarios cuya biografía contenga la palabra 'edad'
10. Obtener el primer usuario cuyo estado sea 'DISPONIBLE'.
11. Buscar todos los usuarios cuyo estado sea 'NO DISPONIBLE'
12. Poned el estado de todos los usuario a 'DISPONIBLE'

Realizar las siguientes pruebas:

1. Insertar el usuario con nombre Pepe Ruiz y correo ppruiz@gmail.com. Con biografia 'perfildepepe' y estado DISPONIBLE.
2. Insertar el usuario con nombre Andrés Ramírez y correo aramirez@gmail.com. Con biografia 'perfildeandres' y estado NO DISPONIBLE
3. Mostrar la lista de todos los usuarios
4. Obtener los datos del usuario con el id de Andrés.
5. Dado el id de Pepe Ruiz, actualizar sus datos: correo ppruiz2@gmail.com y estado NO DISPONIBLE
6. Mostrar la lista de todos los usuarios y comprobar la actualización.

7. Obtener el perfil de Andrés.
8. Mostrar la lista de todos los usuarios cuya biografía contenga la palabra 'perfil'.
9. Mostrar el primer usuario disponible.
10. Mostrar todos los usuarios no disponibles
11. Actualizar todos los usuarios a disponibles.
12. Eliminar el usuario de Pepe.
13. Actualizar el perfil de Pepe usando el método 6: Con biografia 'perfildeandresModificado' y estado NO DISPONIBLE.

Ejercicio 9: Repetir el ejercicio anterior con la relación bidireccional.

Ejercicio 10: Se debe desarrollar una aplicación que permita gestionar la relación entre empleados y oficinas en una empresa, utilizando Hibernate (EntityManager) para la persistencia de datos con una base de datos MariaDB. Una oficina puede tener muchos empleados, pero cada empleado pertenece solo a una oficina. Unidireccional de oficina a empleado. La aplicación debe seguir una arquitectura en capas que incluya modelo, repositorio, servicio y controlador, con los datos gestionados mediante EntityManager. Realizar sólo pruebas en el método run de la clase principal implementando CommandLineRunner. Usar interfaces.

CREATE TABLE `**empleado**` (`id` INT NOT NULL AUTO_INCREMENT, `nombre` VARCHAR(100) NOT NULL, `puesto` VARCHAR(100) NOT NULL, `email` VARCHAR(100) NOT NULL, `oficina_id` INT, PRIMARY KEY (`id`), FOREIGN KEY (`oficina_id`) REFERENCES `oficina`(`id`) ON DELETE SET NULL);	CREATE TABLE `**oficina**` (`id` INT NOT NULL AUTO_INCREMENT, `ubicacion` VARCHAR(255) NOT NULL, `telefono` VARCHAR(20) NOT NULL, PRIMARY KEY (`id`));

Las operaciones a implementar son:
Empleado:

- Crear un empleado
- Consultar todos los empleados.
- Consultar un empleado por su id.
- Actualizar la información de un empleado, incluyendo la oficina asociada.
- Eliminar un empleado por su id.
- Devolver todos los empleados que tengan un puesto especifico.

Oficina:

- Crear una oficina.
- Consultar todas las oficinas.
- Consultar una oficina por su id.
- Eliminar una oficina por su id.
- Contar el número de empleados de una oficinas

- Devolver un mapa donde la clave es el id de la oficina y el valor el número de empleados que tiene esa oficina.
- Devolver el listado de oficinas de más de N empleados.
- Actualiza el teléfono (sólo) de la oficina de un empleado, dado su id.

Haced las pruebas necesarias para comprobar su correcto funcionamiento.

NOTA: Intentad crear clases separadas para las oficinas y los empleados tanto para el repositorio, servicio y controlador.

Ejercicio 11: Repetir el ejercicio anterior con la relación bidireccional.

Ejercicio 12: Desarrollar una aplicación en Spring Boot que gestione una relación One-to-Many unidireccional entre las entidades Autor y Libro. (Un autor tiene muchos libros). Utilizar Hibernate para la persistencia de datos. Implementar el servicio y repositorio de manera independiente. Realizar las pruebas en el archivo Application.java.

CREATE TABLE autor (id INT NOT NULL AUTO_INCREMENT, nombre VARCHAR(255) NOT NULL, PRIMARY KEY (id));	CREATE TABLE libro (id INT NOT NULL AUTO_INCREMENT, titulo VARCHAR(255) NOT NULL, autor_id INT, PRIMARY KEY (id), FOREIGN KEY (autor_id) REFERENCES autor(id));

Métodos para el Servicio:

1. Obtener la lista de todos los autores.
2. Obtener los datos de un autor dado su ID.
3. Insertar un nuevo autor.
4. Actualizar los datos de un autor.
5. Eliminar un autor dado su ID.
6. Añadir un libro a un autor existente.
7. Eliminar un libro de un autor.
8. Obtener todos los libros de un autor dado su ID.
9. Actualizar el título de un libro dado su ID.
10. Obtener la lista de todos los libros.
11. Obtener los autores cuyo nombre contenga una cadena específica.
12. Actualizar el nombre de todos los autores que tengan un libro con un título específico.

Pruebas:

1. Insertar dos autores
2. Insertar un libro para cada autor.
3. Mostrar todos los autores
4. Mostrar todos los libros
5. Obtener el primer autor (por su id)
6. Actualizar el nombre del primer autor
7. Mostrar el actor anterior actualizado

8. Buscar autores cuyo nombre contiene RowLing.
9. Actualizar el título del segundo libro
10. Mostrar los libros.
11. Eliminar el autor segundo
12. Mostrar todos los autores

Ejercicio 13: Repite el ejercicio anterior si la relación del ejercicio anterior es bidireccional.

Ejercicio 14: Desarrollar una aplicación en Spring Boot que gestione una relación One-to-Many bidireccional entre las entidades Curso y Estudiante. (Un curso puede tener muchos estudiantes, pero cada estudiante pertenece a un solo curso). Implementar el servicio y repositorio de manera independiente.

CREATE TABLE curso (id INT NOT NULL AUTO_INCREMENT, nombre VARCHAR(255) NOT NULL, descripcion VARCHAR(500), PRIMARY KEY (id));	CREATE TABLE estudiante (id INT NOT NULL AUTO_INCREMENT, nombre VARCHAR(255) NOT NULL, email VARCHAR(255) NOT NULL, curso_id INT, PRIMARY KEY (id), FOREIGN KEY (curso_id) REFERENCES curso(id));

Métodos para el Servicio:

Curso:

1. Crear un curso con o sin estudiantes.
2. Agregar un estudiante a un curso.
3. Consultar todos los cursos.
4. Consultar un curso por su ID
5. Buscar los cursos cuyo nombre contenga una palabra.
6. Eliminar un estudiante de un curso.
7. Eliminar un curso dado su ID.

Estudiante:

1. Consultar todos los estudiantes.
2. Consultar un estudiante por su ID.
3. Actualizar el email de un estudiante dado su ID.
4. Buscar estudiantes cuyo nombre contenga una cadena específica.

Ejercicio 15: Repetir el ejercicio anterior, pero teniendo en cuenta que un estudiante tiene muchos cursos asignados y un curso puede tener a varios estudiantes inscritos. Hacer la relación

bidireccional.

Ejercicio 16: Un deportista puede practicar muchos deportes y un deporte puede ser practicado por varios deportistas. Desarrollar una aplicación unidireccional desde Deportista hacia Deporte. (Es decir, conociendo el deportista, sabemos la lista de deportes que practica, pero no al revés). Cada deportista tiene un id, y un nombre. Cada deporte tiene un id, y un nombre.
Usar Set en lugar de List en la relación. Realizar los métodos necesarios en el servicio y repositorio para poder probar las siguientes operaciones:

1. Crear dos deportistas: deportista1 y deportista2.
2. Crear 3 deportes: futbol, tenis, y baloncesto
3. Imprimir todos los deportistas. ¿Tienen deportes asociados?
4. Agregar al deportista1 el deporte futbol.
5. Agregar al deportista1 el deporte baloncesto.
6. Agregar al deportista2 el deporte tenis.
7. Imprimir todos los deportistas de nuevo. ¿Tienen deportes asociados?
8. Añadir un nuevo deporte: badminton, al deportista 2.
9. Obtener los datos del deportista2.
10. Eliminar la relación del futbol con el deportista 1.
11. Obtener los datos del deportista1.

Ejercicio 17: ¿Qué cambios se deben aplicar al modelo para que la relación sea bidireccional?

Ejercicio 18. Desarrollar una aplicación tal que:

1. Cada **Persona** tiene un único **Pasaporte** y cada **Pasaporte** pertenece a una sola **Persona**.
2. Una **Persona** puede participar en varios **Proyectos**, y un **Proyecto** puede tener varias **Personas** involucradas.

Todas las relaciones son bidireccionales.

De cada Persona conocemos su id y su nombre. Del Pasaporte conocemos su id y su número. De cada proyecto conocemos su id y su nombre.

Crear en el método run las siguientes pruebas creando en el servicio y repositorio lo necesario. Crear sólo un servicio y un repositorio.

1. Almacenar en BD una persona1 con nombre ‘Juan Perez’.
2. Almacenar en BD una persona2 con nombre ‘Ana Lopez’.
3. Almacenar en BD un pasaporte1 con número ‘ABC123 ‘
4. Almacenar en BD un pasaporte2 con número ‘XYZ456 ‘
5. Almacenar en BD un proyecto1 con nombre con número ‘Proyecto Alpha ‘
6. Almacenar en BD un proyecto2 con nombre con número ‘Proyecto Beta ‘
7. Asignar el pasaporte1 a la persona1
8. Asignar el pasaporte2 a la persona2
9. Asignar el proyecto1 a la persona1
10. Asignar el proyecto2 a la persona1

11. Asignar el proyecto1 a la persona2
12. Obtener todas el nombre de todas las personas con los nombres de sus proyectos.
13. Eliminar el proyecto2 de la persona1