Actividad de recuperación:
Gestión de supermercado con Hibernate.

En esta actividad van a desarrollar una aplicación Java que gestiona una base de datos de un supermercado utilizando Hibernate como framework ORM. Implementarán el mapeo objeto-relacional mediante anotaciones JPA y realizarán operaciones CRUD completas sobre las entidades, prestando especial atención a las relaciones entre tablas y las operaciones en cascada.

Modelo de datos:

El sistema de gestión de supermercado debe permitir administrar proveedores, productos y lotes de stock. El modelo de datos está compuesto por tres entidades principales:

- **Proveedor**: Representa a las empresas que suministran productos.
- **Producto**: Representa los artículos disponibles para la venta.
- **Lote**: Representa las remesas de stock de cada producto con fecha de caducidad.

Relaciones:

- Un proveedor puede suministrar múltiples productos (1:N).
- Un producto es suministrado por un único proveedor (N:1).
- Un producto puede tener múltiples lotes de stock (1:N).
- Un lote pertenece a un único producto (N:1).

Operaciones en cascada:

- Al eliminar un proveedor, se deben eliminar todos sus productos.
- Al eliminar un producto, se deben eliminar todos sus lotes.
- Al guardar un proveedor con productos nuevos, los productos deben guardarse automáticamente.

Apartado 1. Preparación del entorno.

1.1. Creación de la base de datos.

Ejecuten el script SQL `supermercado.sql` en MySQL para crear la base de datos `supermercado_hibernate`.

1.2. Configuración del proyecto Maven.

Creen un proyecto Maven y añadan las dependencias necesarias en el archivo `pom.xml`.

1.3. Configuración de Hibernate.

Creen el archivo `hibernate.cfg.xml` en la carpeta `src/main/resources`.

1.4. Clase HibernateUtil (paquete com.dam.util).

Implementen una clase de utilidad para gestionar el SessionFactory.

Entrega: Captura de pantalla mostrando la base de datos creada con datos y la estructura del proyecto Maven.

Apartado 2. Implementación de entidades.

Implementen las tres clases de entidad utilizando anotaciones JPA. Sigan las especificaciones indicadas:

Clase Proveedor (paquete com.dam.modelo).

- Mapear a la tabla `proveedores`.
- Definir `id_proveedor` como clave primaria autogenerada.
- Establecer relación OneToMany con Producto.
- Configurar cascada `CascadeType.ALL` y `orphanRemoval = true`.
- Implementar métodos helper: `addProducto()` y `removeProducto()` para mantener la consistencia bidireccional.
- Incluir constructor sin parámetros y constructor con todos los campos excepto ID.
- Sobrescribir `toString()` mostrando información relevante.

Clase Producto (paquete com.dam.modelo).

- Mapear a la tabla `productos`.
- Definir `id_producto` como clave primaria autogenerada.
- Establecer relación ManyToOne con Proveedor (`FetchType.LAZY`).
- Establecer relación OneToMany con Lote.
- Configurar cascada `CascadeType.ALL` y `orphanRemoval = true` en la relación con Lote.
- Implementar métodos helper: `addLote()` y `removeLote()`.
- Incluir constructores y sobrescribir `toString()`.

Clase Lote (paquete com.dam.modelo).

- Mapear a la tabla `lotes`.
- Definir `id_lote` como clave primaria autogenerada.

- Crear un enum `EstadoLote` con los valores: DISPONIBLE, VENDIDO, CADUCADO, RETIRADO.
- Mapear el campo estado usando `@Enumerated(EnumType.STRING)`.
- Establecer relación ManyToOne con Producto (`FetchType.LAZY`).
- Incluir constructores y sobrescribir `toString()`.

Entrega: Código fuente de las tres clases (Proveedor, Producto, Lote) correctamente implementadas y comentadas.

Apartado 3: Operaciones Create.

Creen la clase `GestionSupermercado` (paquete `com.dam.gestion`) e implementen los siguientes métodos:

Método crearProveedorConProductos().

Creen un nuevo proveedor con los siguientes datos:

- Nombre: "Distribuciones García S.L."
- CIF: "B-12345678"
- Teléfono: "918765432"
- Email: "contacto@distgarcia.es"

Asocien al proveedor DOS productos nuevos, cada uno con los siguientes datos:

Producto 1:

- Nombre: "Aceite de oliva virgen extra"
- Código de barras: "8412345678901"
- Precio: 8.95
- Categoría: "Alimentación"
- Dos lotes:
    - Código: "LOT-2024-001", Cantidad: 50, Fecha caducidad: 31/12/2025, Estado: DISPONIBLE
    - Código: "LOT-2024-002", Cantidad: 30, Fecha caducidad: 31/12/2025, Estado: DISPONIBLE

Producto 2:

- Nombre: "Arroz integral 1kg"
- Código de barras: "8412345678902"
- Precio: 2.45
- Categoría: "Alimentación"

- Un lote:
    - Código: "LOT-2024-003", Cantidad: 100, Fecha caducidad: 30/06/2026, Estado: DISPONIBLE

Importante: Utilicen las operaciones en cascada para guardar todo con una sola operación `session.persist(proveedor)`.

Método agregarLoteAProductoExistente(String codigoBarras, String codigoLote).

Busquen un producto por su código de barras y añádanle un nuevo lote con el código proporcionado.

El lote debe tener:

- Cantidad: 25
- Estado: DISPONIBLE
- Fecha de caducidad: un año desde la fecha actual

Deben gestionar el caso en que el producto no exista mostrando un mensaje apropiado.

Entrega:

- Código fuente de los dos métodos implementados.
- Captura de pantalla de la consola mostrando las sentencias SQL generadas.
- Captura de la base de datos mostrando los nuevos registros insertados.

Apartado 4: Operaciones Read.

Implementen los siguientes métodos de consulta en la clase `GestionSupermercado`:

Método listarTodosLosProveedores().

Recupera todos los proveedores de la base de datos y muestra por consola:

- ID del proveedor.
- Nombre de la empresa.
- CIF.
- Número de productos que suministra.

Utiliza una consulta HQL y recorre la lista mostrando la información.

Método buscarProductoPorId(Integer idProducto).

Busca un producto por su ID y muestra toda su información:

- Datos del producto (nombre, código de barras, precio, categoría).
- Datos del proveedor.
- Lista de todos sus lotes con cantidad, fecha de caducidad y estado.

Gestiona el caso en que el producto no exista.

Método buscarLotesPorEstado(EstadoLote estado).

Utiliza HQL para buscar todos los lotes que tengan un estado determinado. La consulta HQL debe ser: `"FROM Lote l WHERE l.estado = :estado"`.

Muestra por consola:

- Código del lote
- Cantidad disponible
- Estado
- Fecha de caducidad
- Nombre del producto al que pertenece

Método obtenerEstadisticasSupermercado().

Muestra las siguientes estadísticas:

- Número total de proveedores
- Número total de productos
- Número total de lotes
- Número de lotes por cada estado (DISPONIBLE, VENDIDO, etc.)

Utiliza consultas HQL con funciones de agregación: `"SELECT COUNT(l) FROM Lote l WHERE l.estado = :estado"`.

Entrega:

- Código fuente de los cuatro métodos
- Captura de pantalla de la salida por consola de cada método

---

Apartado 5: Operaciones Update.

Implementen los siguientes métodos de actualización:

Método actualizarEstadoLote(Integer idLote, EstadoLote nuevoEstado).

Busca un lote por su ID y actualiza su estado. Debe:

- Comprobar que el lote existe

- Cambiar el estado
- Mostrar un mensaje confirmando el cambio
- Si el nuevo estado es VENDIDO, cambiar la cantidad a 0

Método actualizarPrecioProducto(Integer idProducto, Double nuevoPrecio).

Busca un producto por su ID y actualiza su precio. Debe gestionar que el producto exista antes de modificarlo y validar que el precio sea positivo.

Método transferirProductosEntreProveedores(Integer idProveedorOrigen, Integer idProveedorDestino).

Transfiere todos los productos de un proveedor a otro proveedor. Debe:

- Verificar que ambos proveedores existen
- Obtener todos los productos del proveedor origen
- Cambiar el proveedor de cada producto al proveedor destino
- Mantener la consistencia de las relaciones bidireccionales

Entrega:

- Código fuente de los tres métodos.
- Capturas de pantalla mostrando:
    - Datos antes de la actualización
    - Ejecución del método
    - Datos después de la actualización

Apartado 6: Operaciones Delete.

Implementen los siguientes métodos de eliminación:

Método eliminarLote(Integer idLote).

Elimina un lote específico por su ID. Debe:

- Verificar que el lote existe
- Mantener la consistencia de la relación bidireccional (usar el método `removeLote()` del producto)
- Mostrar un mensaje de confirmación

Método eliminarProducto(Integer idProducto).

Elimina un producto por su ID. Debe:

- Verificar que el producto existe
- Mantener la consistencia de la relación bidireccional con el proveedor
- Mostrar mensaje indicando cuántos lotes se eliminaron en cascada

Método eliminarProveedorConCascada(Integer idProveedor).

Elimina un proveedor y, gracias a la cascada, todos sus productos y lotes. Debe:

- Buscar el proveedor y contar cuántos productos tiene
- Contar el total de lotes de esos productos
- Eliminar el proveedor
- Mostrar un resumen: "Eliminado proveedor X con Y productos y Z lotes"

Importante: Demuestren que las operaciones en cascada funcionan correctamente.

Entrega:

- Código fuente de los tres métodos.
- Capturas de pantalla mostrando:
    - Estado de la BD antes de eliminar
    - Salida por consola al ejecutar los métodos
    - Estado de la BD después de eliminar (mostrando que se eliminaron las entidades relacionadas)

Apartado 7: Clase principal y menú.

Creen una clase `Main` con un método `main` que implemente un menú interactivo por consola con las siguientes opciones:

```
=== SISTEMA DE GESTIÓN DE SUPERMERCADO ===
1. Crear nuevo proveedor con productos
2. Agregar lote a producto existente
3. Listar todos los proveedores
4. Buscar producto por ID
5. Buscar lotes por estado
6. Mostrar estadísticas
7. Actualizar estado de lote
8. Actualizar precio de producto
9. Eliminar lote
10. Eliminar producto
11. Eliminar proveedor (con cascada)
0. Salir

Seleccione una opción:
```

El menú debe:

- Ejecutarse en un bucle hasta que el usuario elija salir
- Solicitar los datos necesarios para cada operación
- Gestionar errores de entrada (InputMismatchException)
- Llamar al método correspondiente de `GestionSupermercado`
- Mostrar mensajes claros al usuario
- Cerrar correctamente el SessionFactory al salir

Entrega:

- Código fuente de la clase Main con el menú completo.
- Vídeo mostrando la ejecución de todas las opciones del menú.

Información adicional.

Estructura del proyecto.

```
SupermercadoHibernate/
├── src/
│   └── main/
│       ├── java/
│       │   └── com/
│       │       └── dam/
│       │           ├── modelo/
│       │           │   ├── Proveedor.java
│       │           │   ├── Producto.java
│       │           │   └── Lote.java
│       │           ├── util/
│       │           │   └── HibernateUtil.java
│       │           ├── gestion/
│       │           │   └── GestionSupermercado.java
│       │           └── Main.java
│       └── resources/
│           └── hibernate.cfg.xml
├── pom.xml
├── memoria.pdf
└── evidencias/
    ├── capturas/
    └── videos/
```

Contenido de la entrega.

1. Proyecto completo comprimido.
2. Documento PDF (memoria.pdf) que incluya:
    - Portada con tus datos
    - Índice
    - Breve explicación del funcionamiento de cada apartado

- Todas las capturas de pantalla solicitadas
- Fragmentos de código relevantes comentados
- Dificultades encontradas y soluciones aplicadas
- Conclusiones personales

Suban al aula virtual un archivo comprimido con todo lo solicitado. El nombre del archivo debe formarse con su nombre y el número de actividad, por ejemplo: "Nombre_Apellido.Recuperacion_Hibernate.zip"

Criterios de calificación.

Esta actividad se calificará de 0 a 10:

- **1'0 puntos. Apartado 1.**
  Base de datos creada correctamente. Configuración Maven y Hibernate correcta.
- **1'5 puntos. Apartado 2.**
  Entidades correctamente mapeadas con anotaciones. Relaciones bidireccionales implementadas. Cascada configurada.
- **1'5 puntos. Apartado 3.**
  Operaciones de inserción funcionando correctamente. Uso correcto de cascada. Gestión de transacciones.
- **1'5 puntos. Apartado 4.**
  Consultas HQL correctas. Recuperación de datos con relaciones. Manejo de colecciones.
- **1'0 puntos. Apartado 5.**
  Actualizaciones funcionando. Consistencia de datos mantenida.
- **1'0 puntos. Apartado 6.**
  Eliminaciones correctas. Demostración de cascada en DELETE. Gestión de relaciones.
- **0'5 puntos. Apartado 7.**
  Menú funcional y completo. Experiencia de usuario adecuada. Cierre correcto de recursos.
- **1'0 puntos. Memoria y documentación.**
  Documento bien estructurado. Explicaciones claras. Capturas adecuadas.
- **1'0 puntos. Calidad del código.**
  Código limpio y organizado. Comentarios adecuados. Manejo de excepciones. Nombres descriptivos.