Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

Actividad: Supermercado.

Enlace al repositorio de la página: GitHub - rodrigo-cyber17

Actividad. Módulo Profesional: Acceso a adatos.

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.

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

ÍNDICE

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

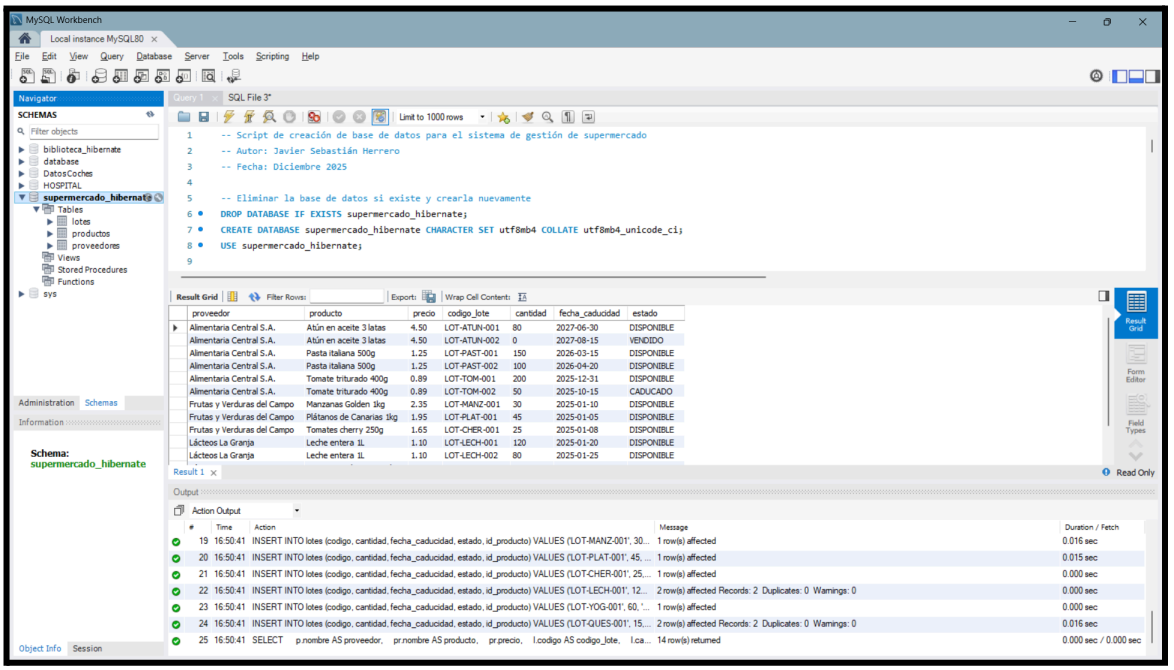
---

1. Paso

"Preparación del entorno".

1.1. Paso

"Creación de la base de datos".

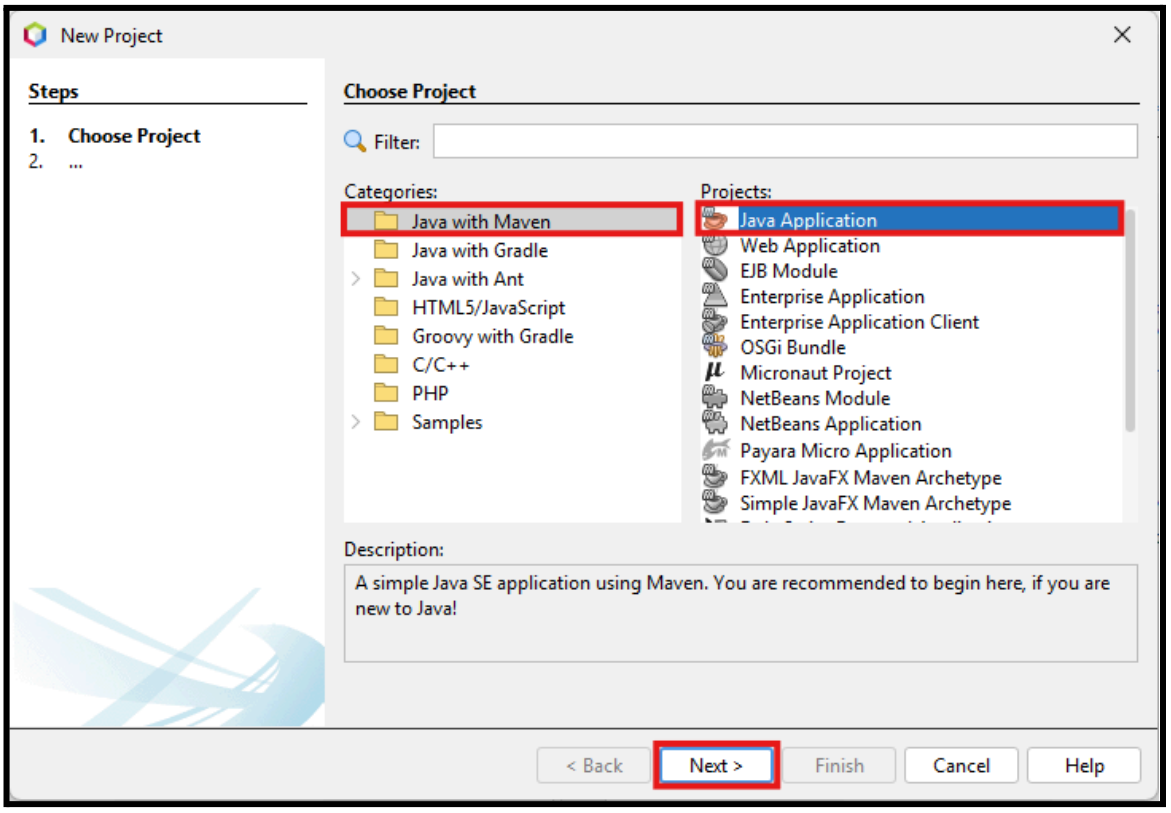


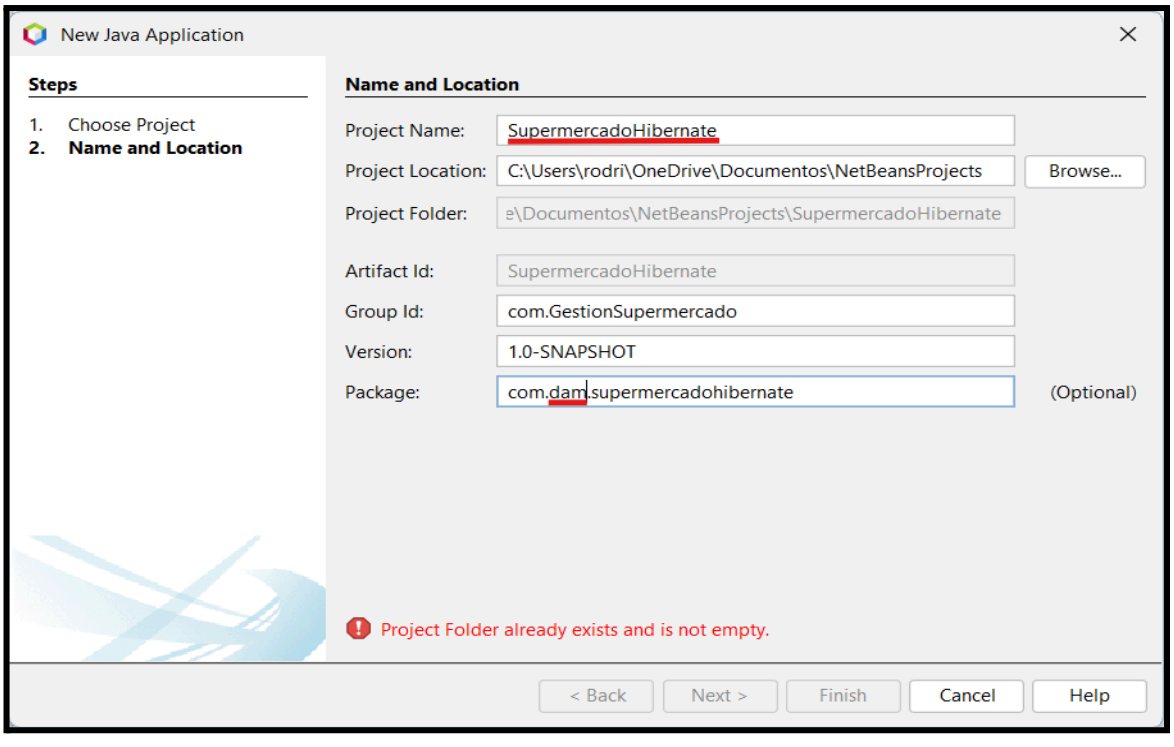
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

1.2. Paso

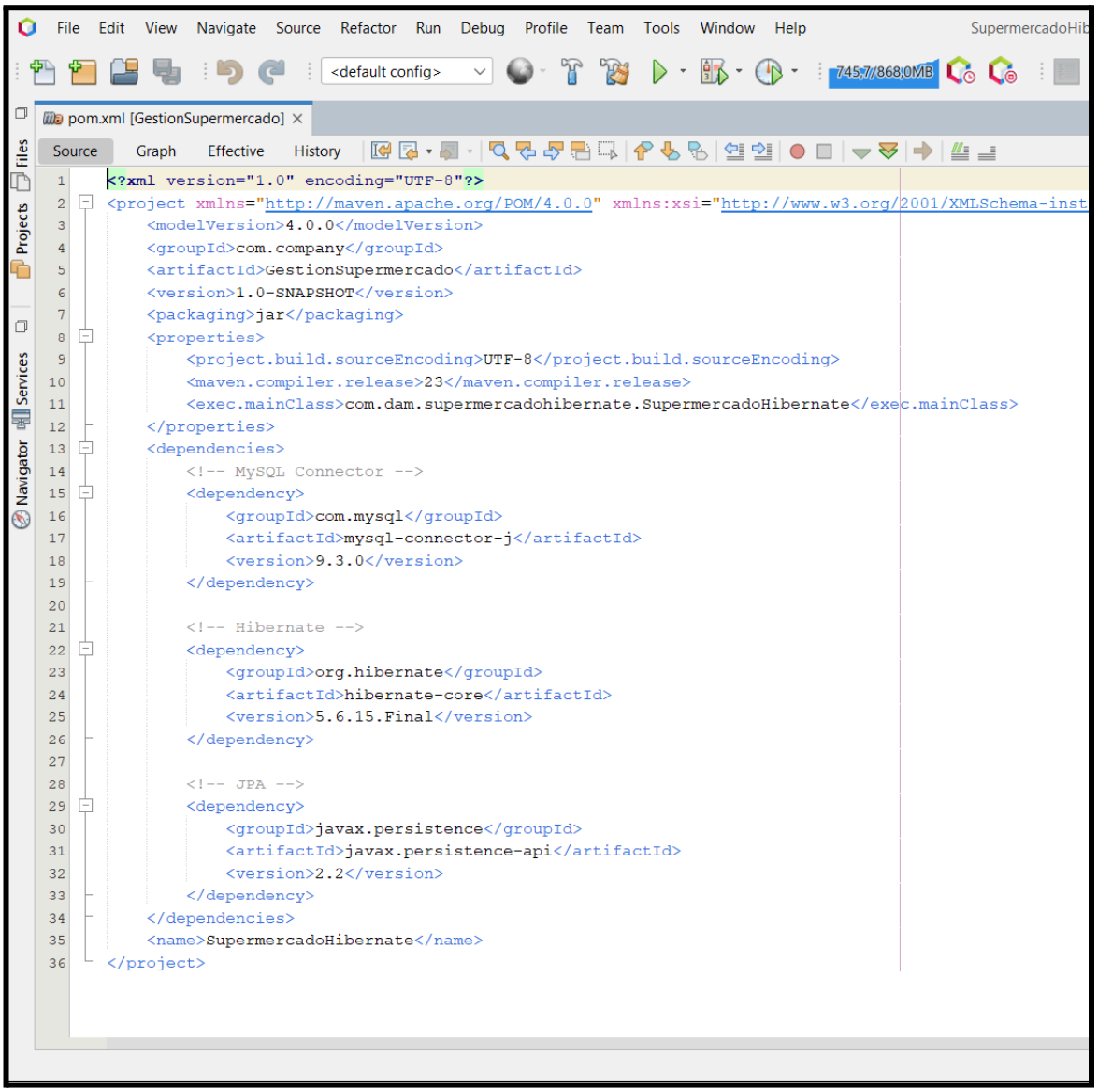
"Configuración del proyecto Maven".

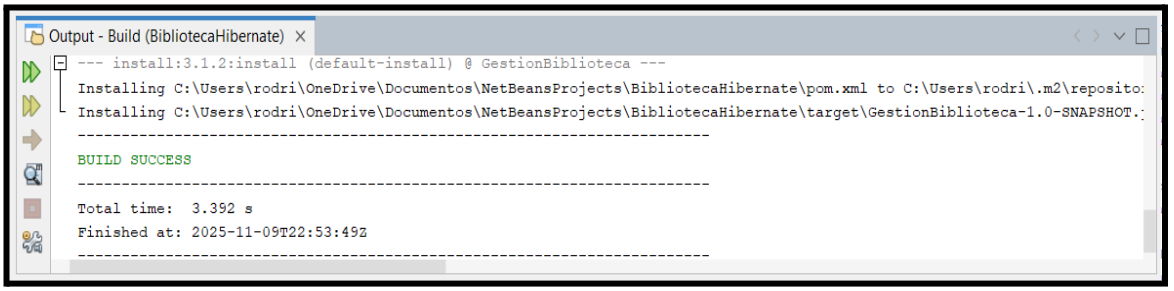




Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)



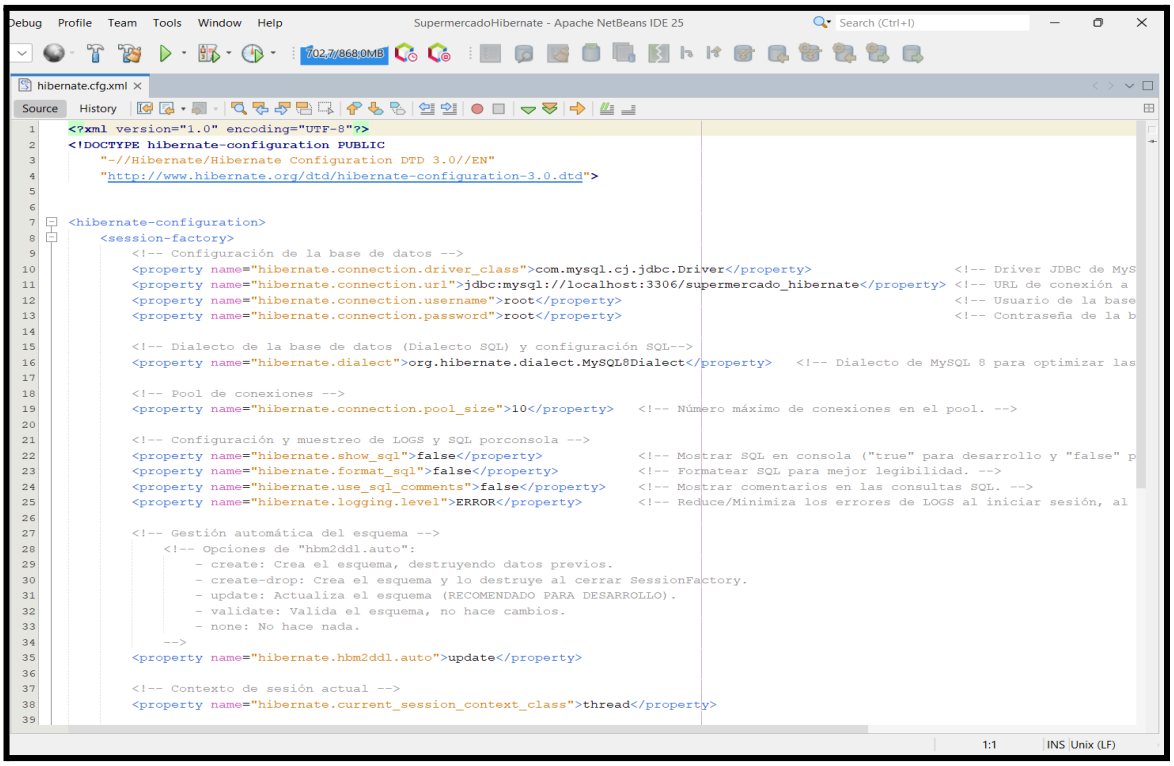


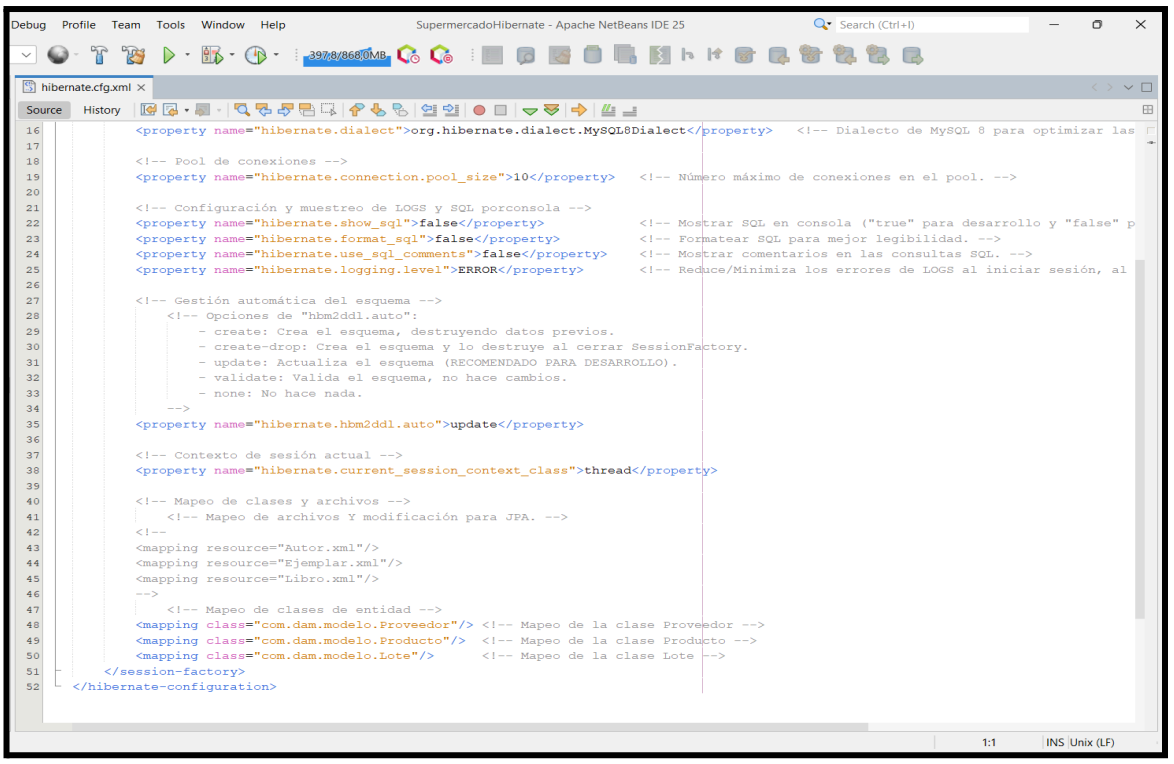
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

1.3. Paso

"Configuración de Hibernate".



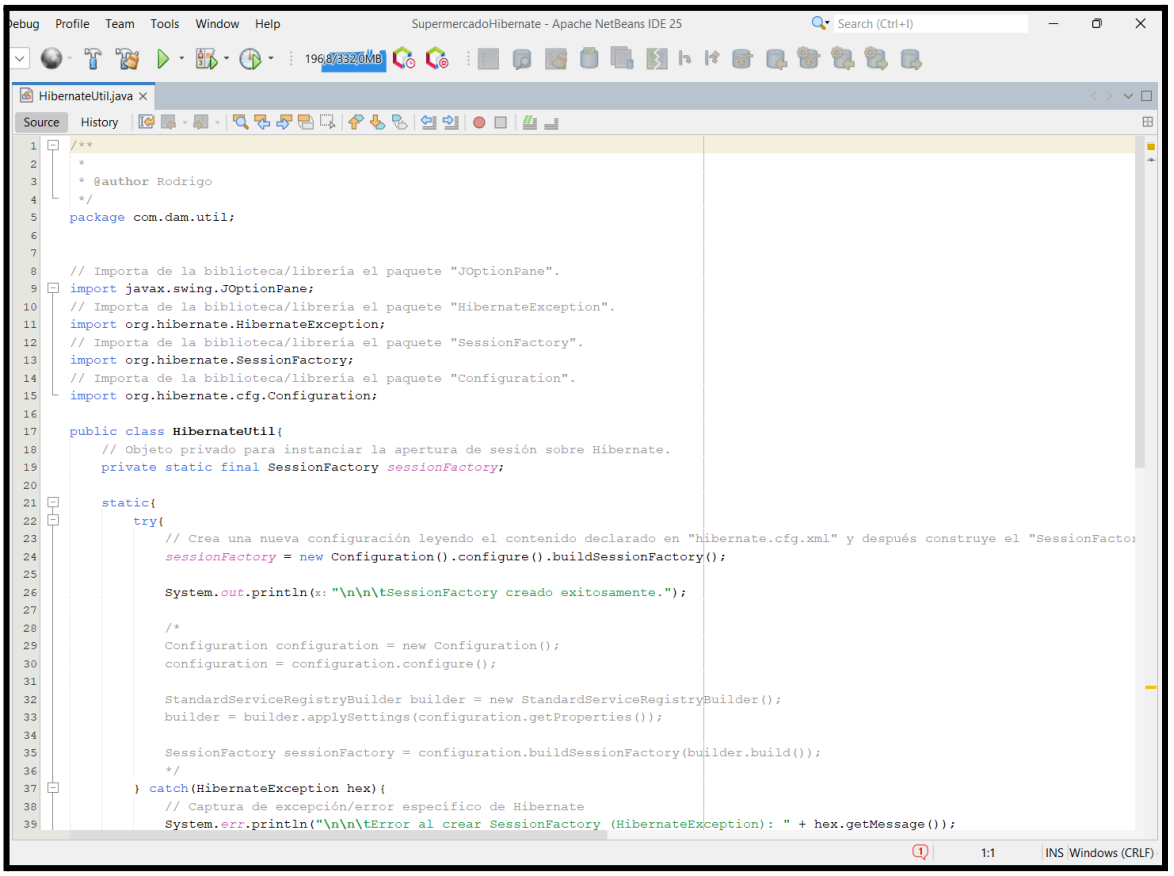


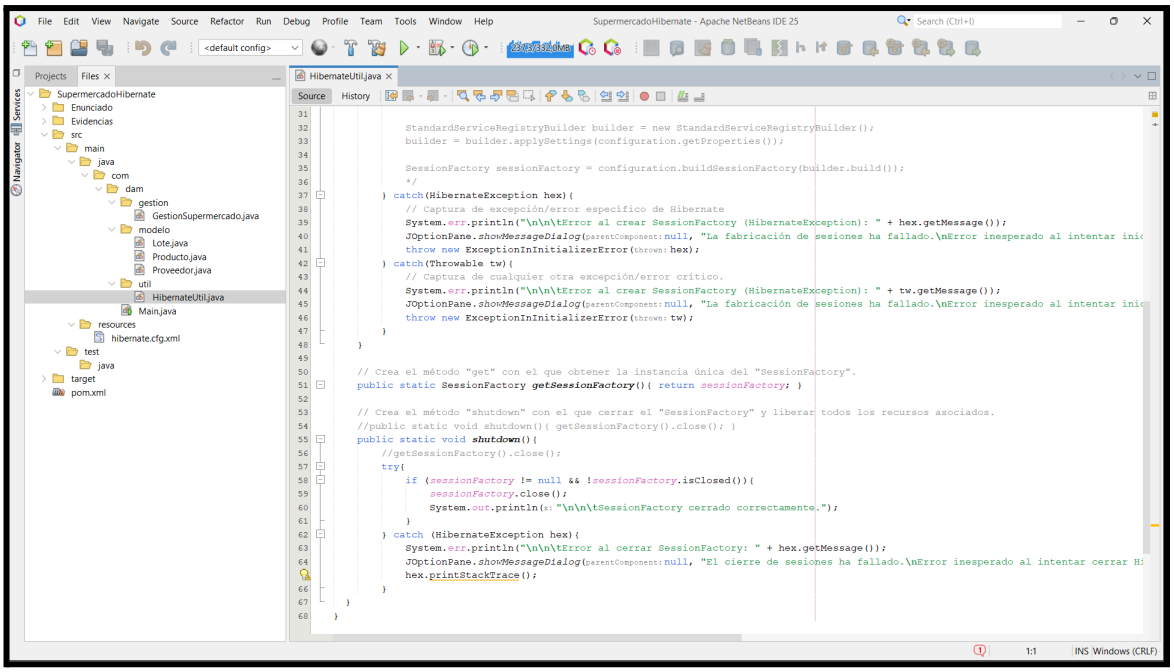
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

1.4. Paso

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

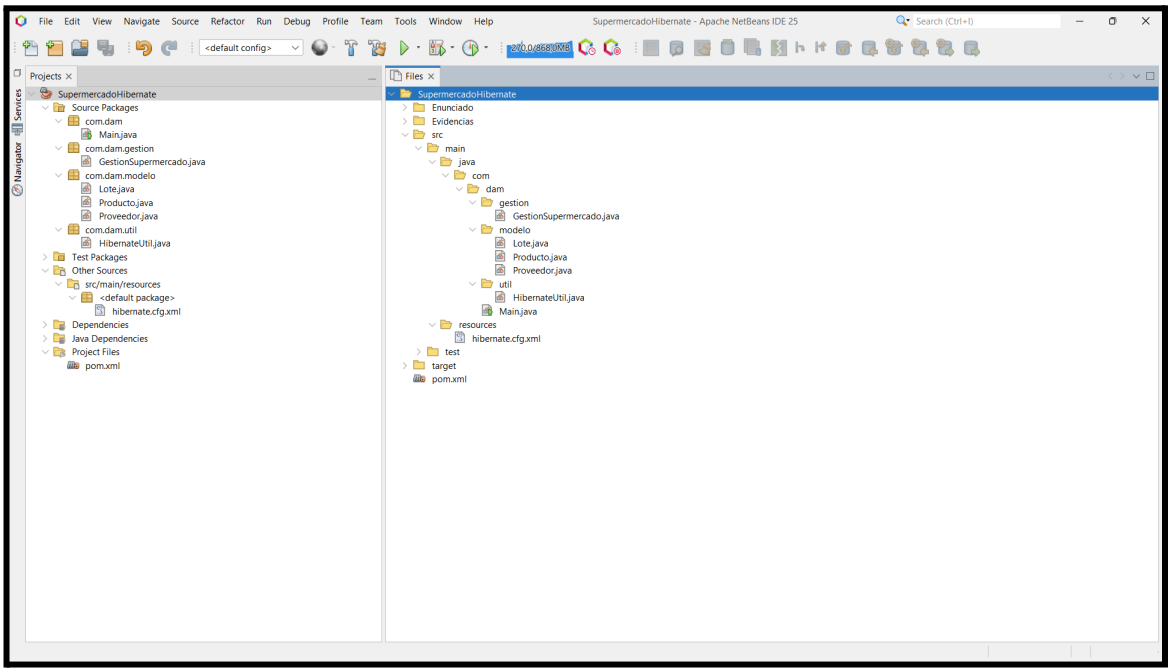




Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

Resultado: Estructuración del proyecto



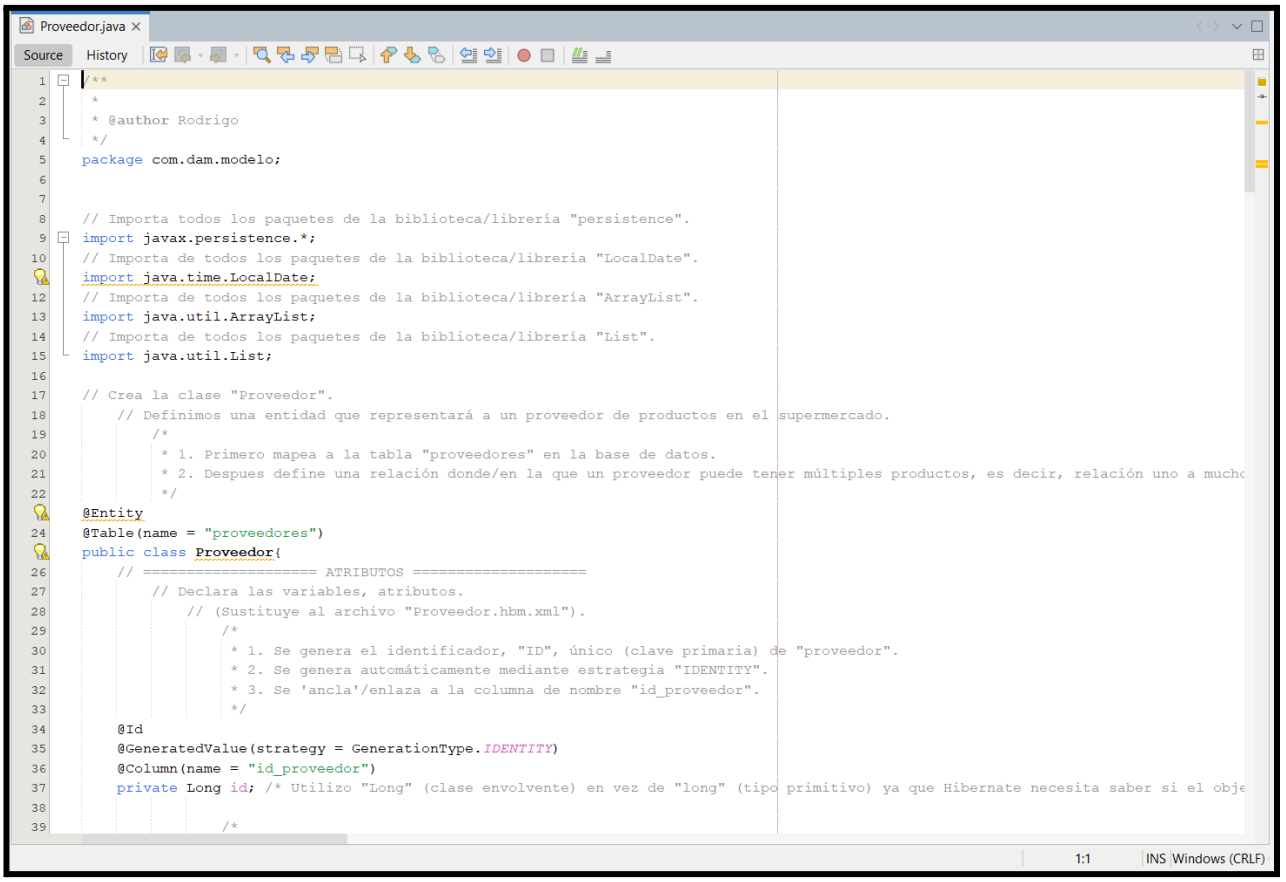
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

---

2. Paso

"Implementación de entidades".



Entidad 'Proveedor' (Proveedor.java).

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

Producto.java

Source | History

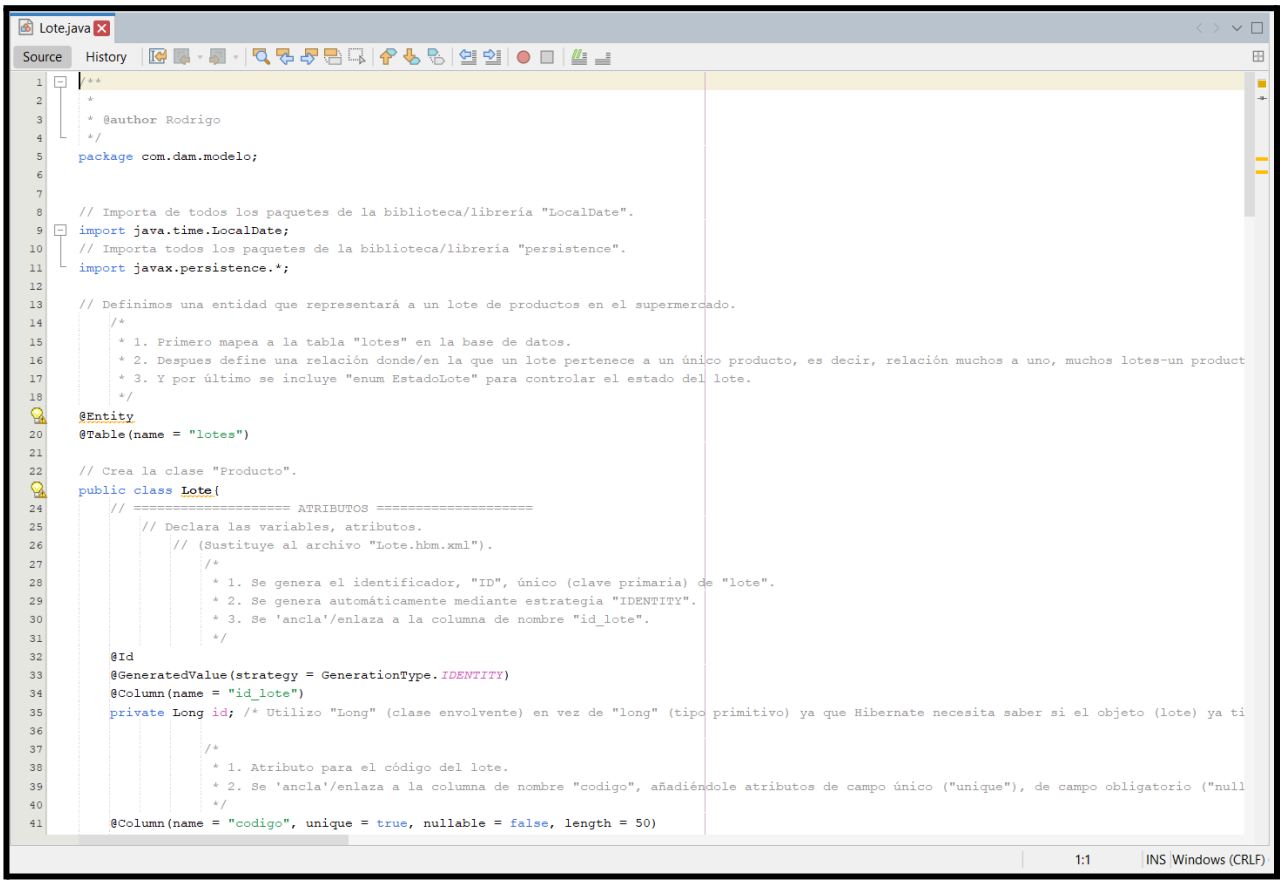
```java
/**
 *
 * @author Rodrigo
 */
package com.dam.modelo;


// Importa todos los paquetes de la biblioteca/librería "persistence".
import javax.persistence.*;
// Importa de todos los paquetes de la biblioteca/librería "ArrayList".
import java.util.ArrayList;
// Importa de todos los paquetes de la biblioteca/librería "List".
import java.util.List;

// Crea la clase "Producto".
    // Definimos una entidad que representará a un producto correspondiente de un lote de estos en el supermercado.
        /*
         * 1. Primero mapea a la tabla "productos" en la base de datos.
         * 2. Despues define una relación donde/en la que múltiples productos corresponden a un único proveedor, es decir, relación muchos a uno, un
         * 3. Y por último define una relación donde/en la que un producto puede corresponder a múltiples lotes, es decir, relación uno a muchos, un
         */
@Entity
@Table(name = "productos")
public class Producto{
    // ================== ATRIBUTOS ====================
        // Declara las variables, atributos.
            // (Sustituye al archivo "Producto.hbm.xml").
                /*
                 * 1. Se genera el identificador, "ID", único (clave primaria) de "producto".
                 * 2. Se genera automáticamente mediante estrategia "IDENTITY".
                 * 3. Se 'ancla'/enlaza a la columna de nombre "id_producto".
                 */
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_producto")
    private Long id; /* Utilizo "Long" (clase envolvente) en vez de "long" (tipo primitivo) ya que Hibernate necesita saber si el objeto (producto) y


                /*
                 * 1. Atributo para nombre del producto.
                 * 2. Se 'ancla'/enlaza a la columna de nombre "nombre", añadiéndole atributos de campo obligatorio ("nullable") y de máximo 200 cara
                 */
```

1:1 INS Windows (CRLF)

Entidad 'Producto' (Producto.java).

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)



Entidad 'Lote' (Lote.java).

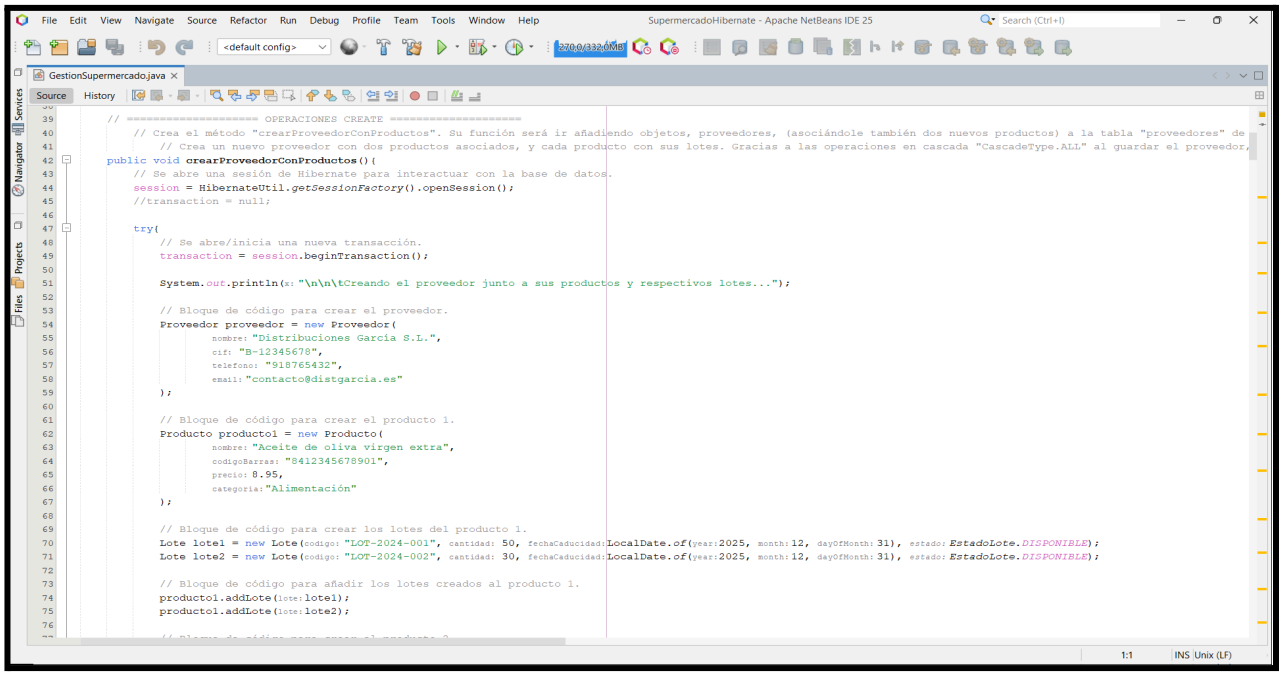
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

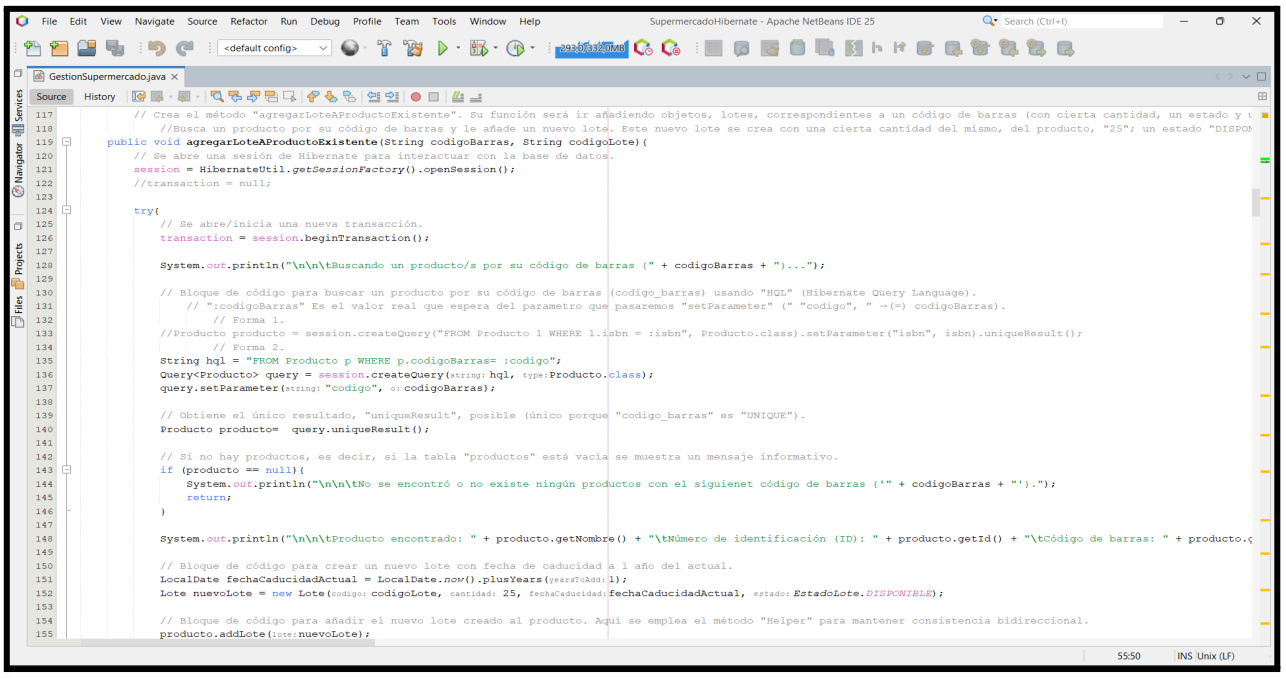
---

3. Paso

"Operaciones Create".



Operación 'crear preoveedor/es' (Método "crearProveedorConProductos()") (GestionSupermercado.java).



Operación 'crear lote/s' (Método "agregarLoteAProductoExistente(String codigoBarras, String codigoLote)") (GestionSupermercado.java).

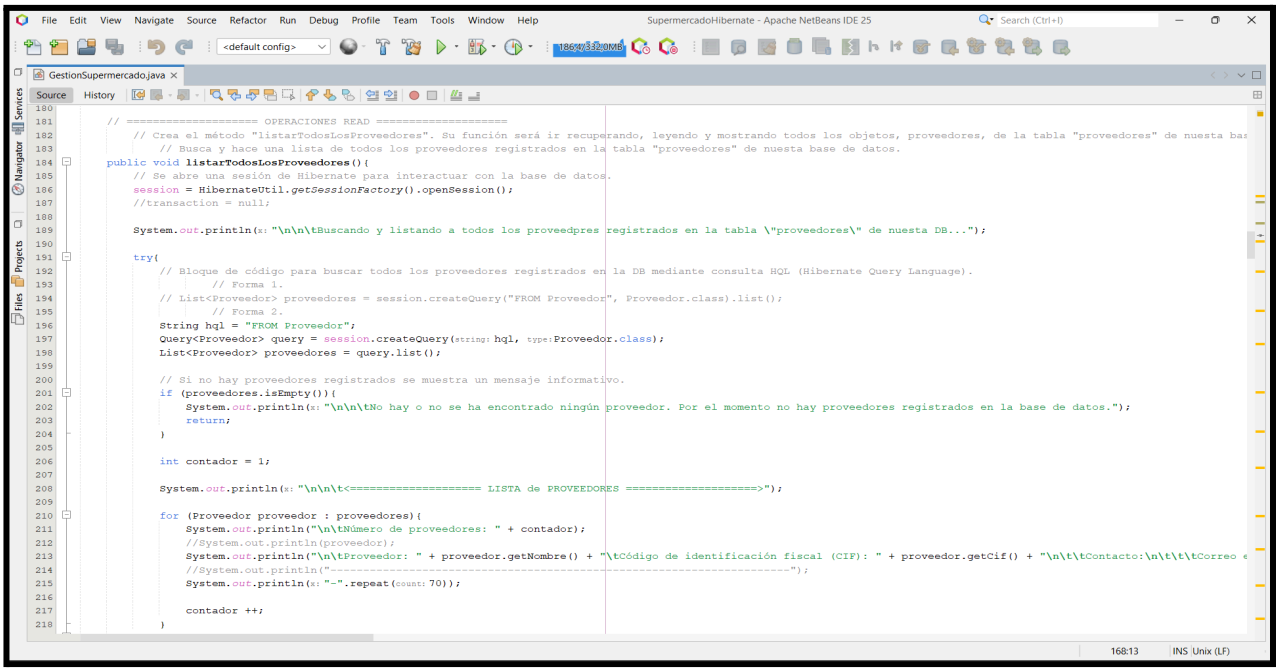
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

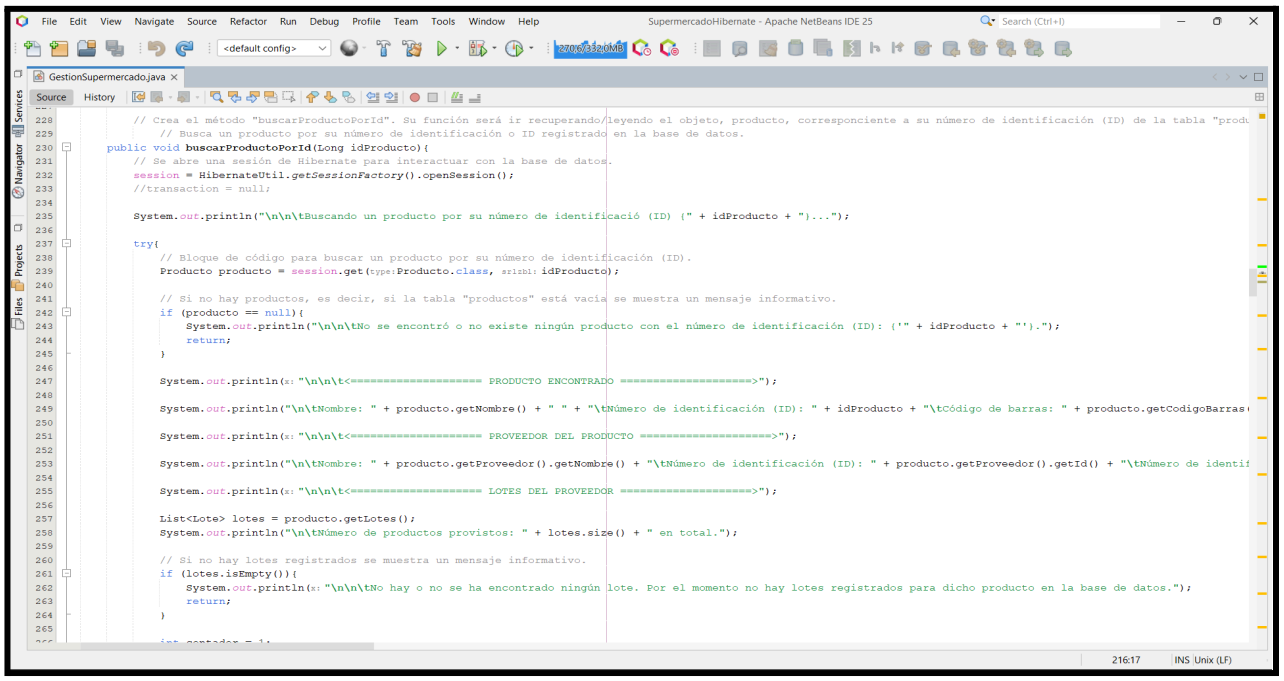
---

4. Paso

"Operaciones Read".



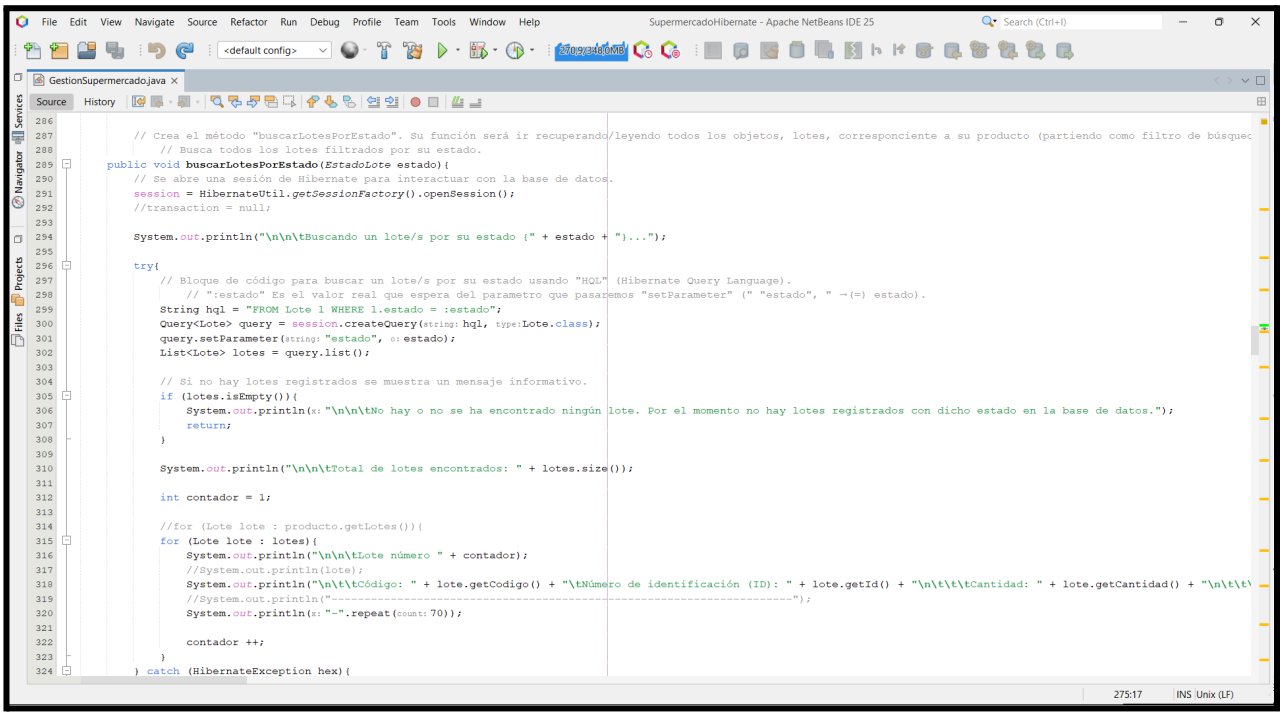
Operación 'listar proveedor/es' (Método "listarTodosLosProveedores()") (GestionSupermercado.java).



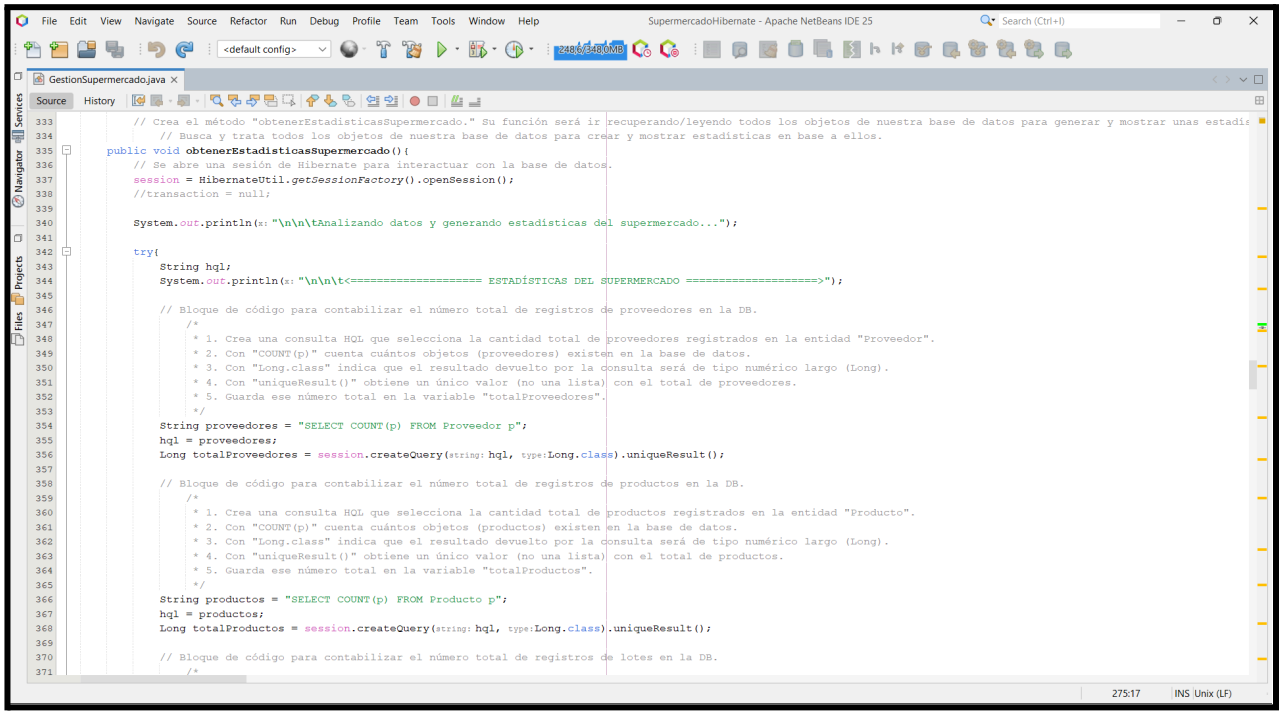
Operación 'buscar producto' (Método "buscarProductoPorId(Long idProducto)") (GestionSupermercado.java).

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)



Operación 'buscar lote/s' (Método "buscarLotesPorEstado(EstadoLote estado)") (GestionSupermercado.java).



Operación 'obtener estadísticas' (Método "obtenerEstadisticasSupermercado()") (GestionSupermercado.java).

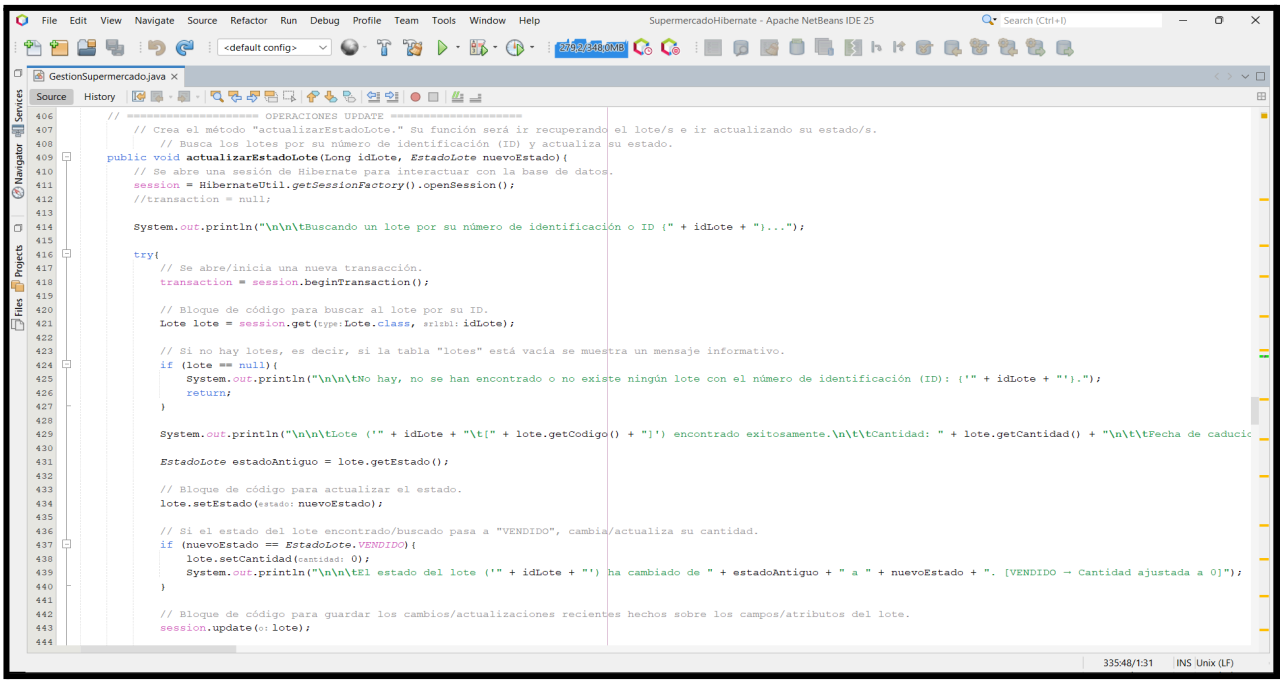
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

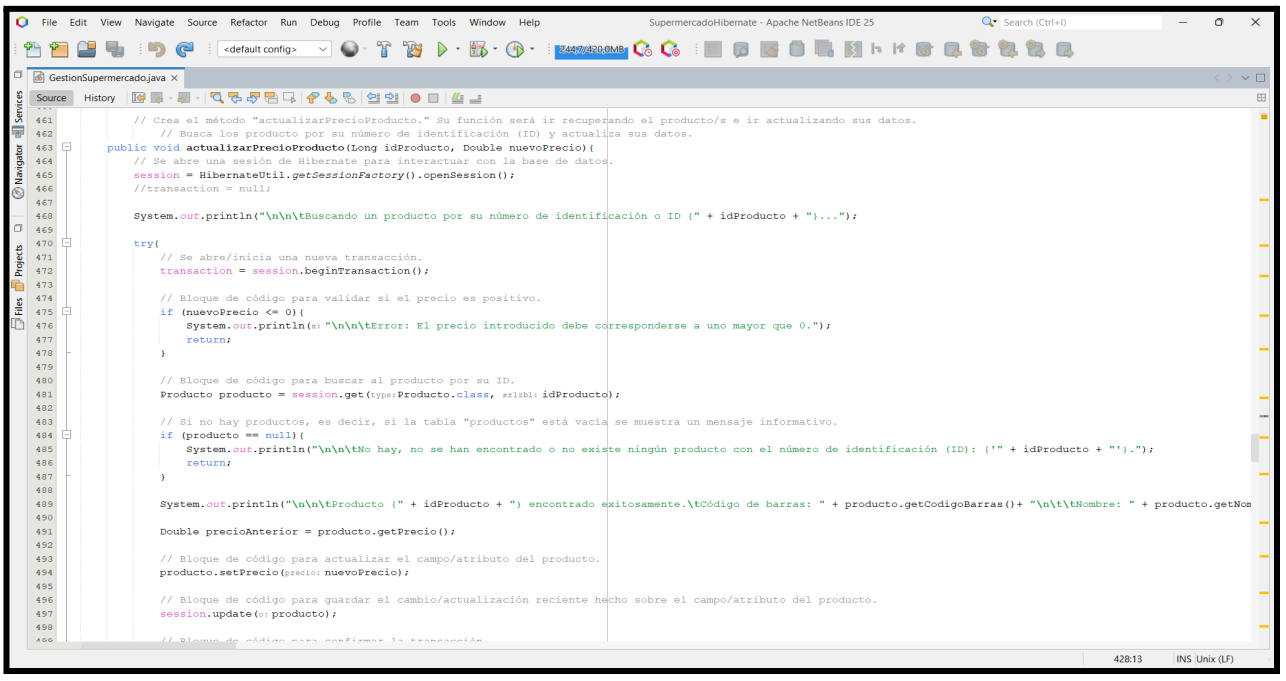
---

5. Paso

"Operaciones Update".



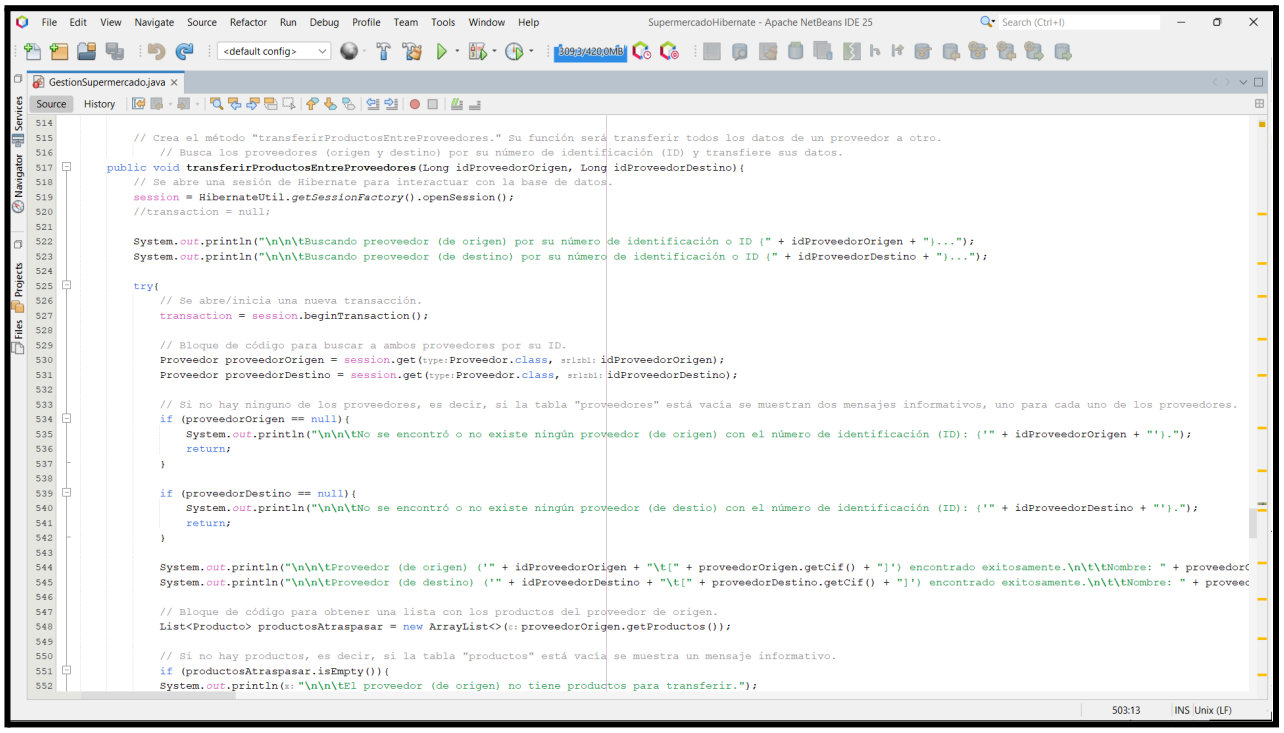
Operación 'actualizar estado lote/s' (Método "actualizarEstadoLote(Long idLote, EstadoLote nuevoEstado)") (GestionSupermercado.java).



Operación 'actualizar precio producto/s' (Método "actualizarPrecioProducto(Long idProducto, Double nuevoPrecio)") (GestionSupermercado.java).

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)



Operación 'transferir productos entre proveedores' (Método "transferirProductosEntreProveedores(Long idProveedorOrigen, Long idProveedorDestino)") (GestionSupermercado.java).

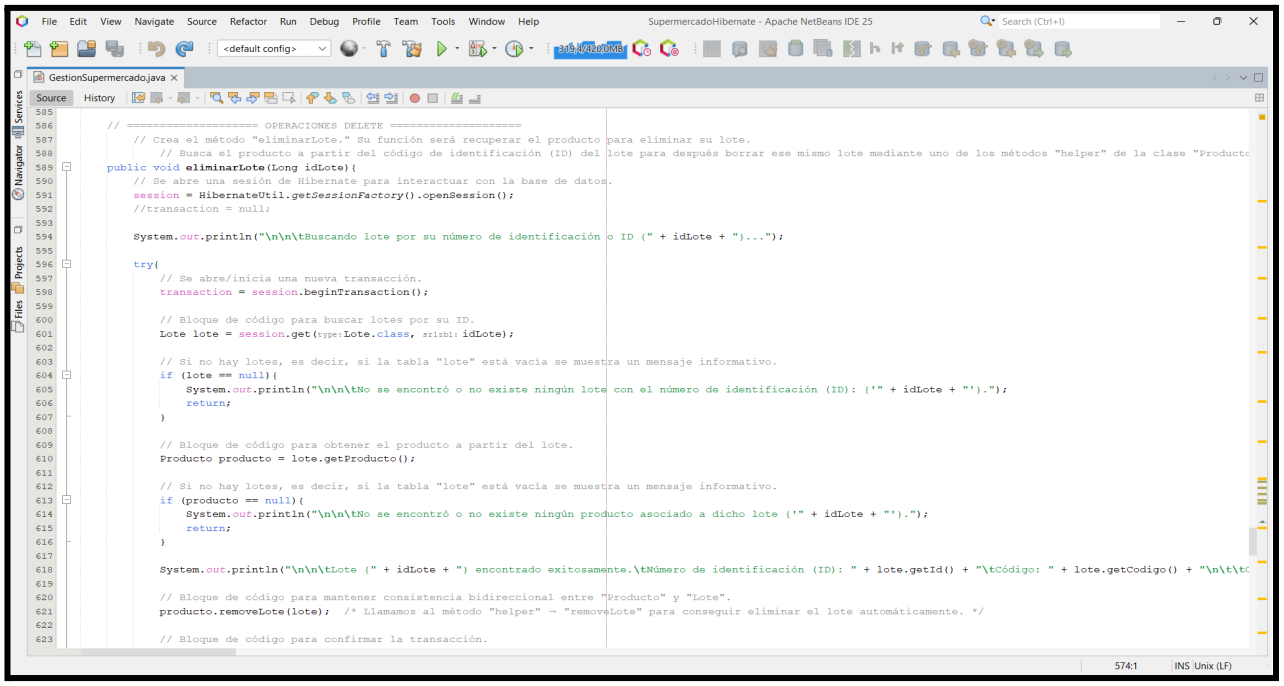
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

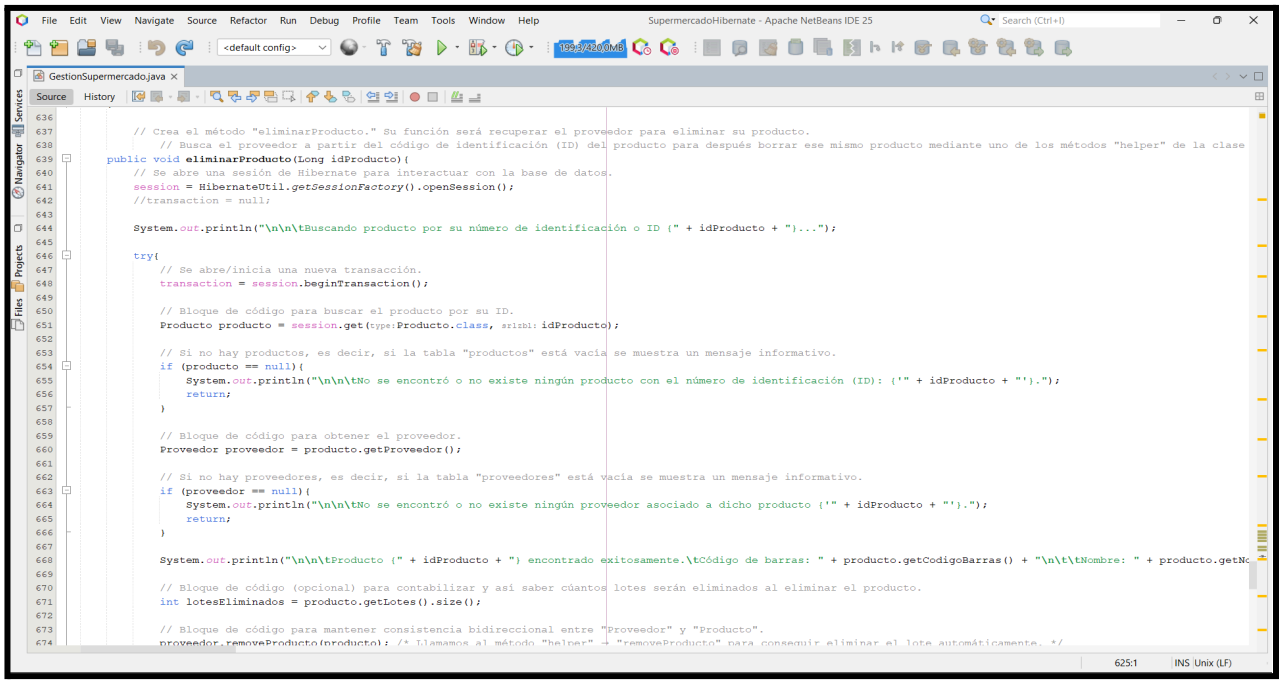
---

6. Paso

"Operaciones Delete".



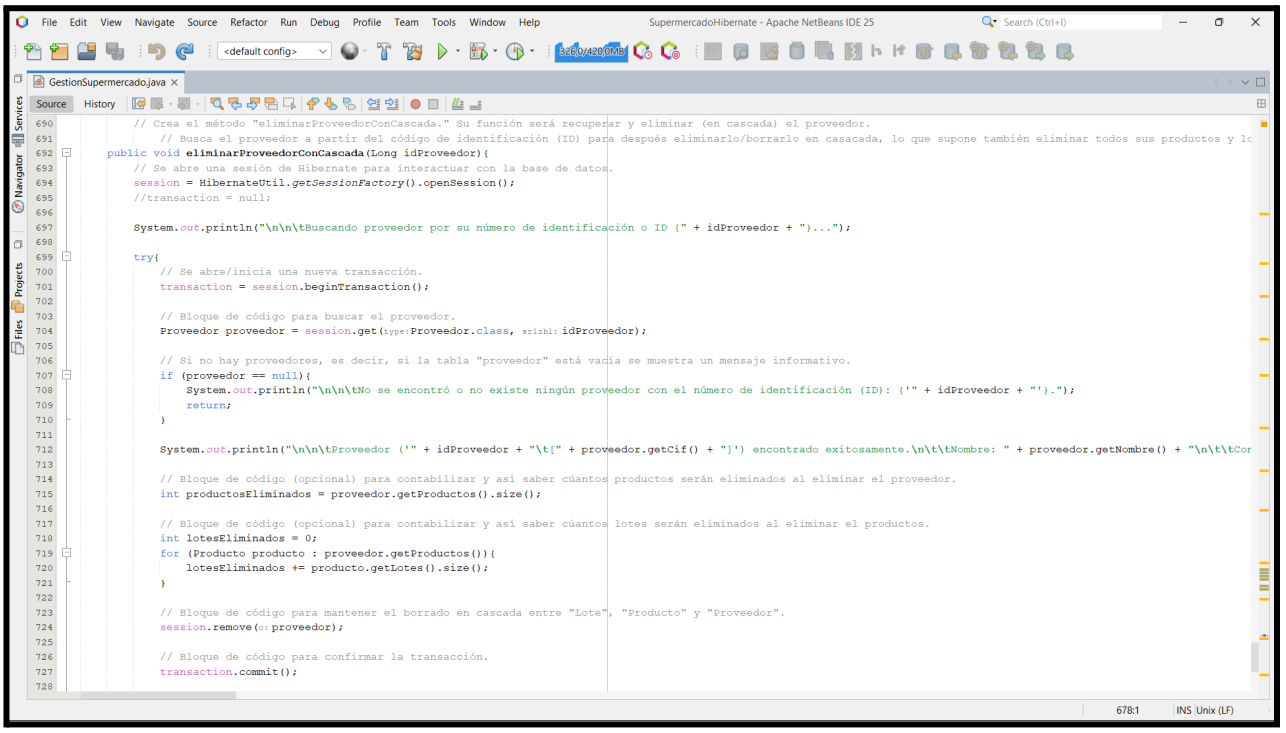
Operación 'eliminar lote/s' (Método "eliminarLote(Long idLote)") (GestionSupermercado.java).



Operación 'eliminar producto/s' (Método "eliminarProducto(Long idProducto)") (GestionSupermercado.java).

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)



Operación 'eliminar proveedor/es' (Método "eliminarProveedorConCascada(Long idProveedor)") (GestionSupermercado.java).

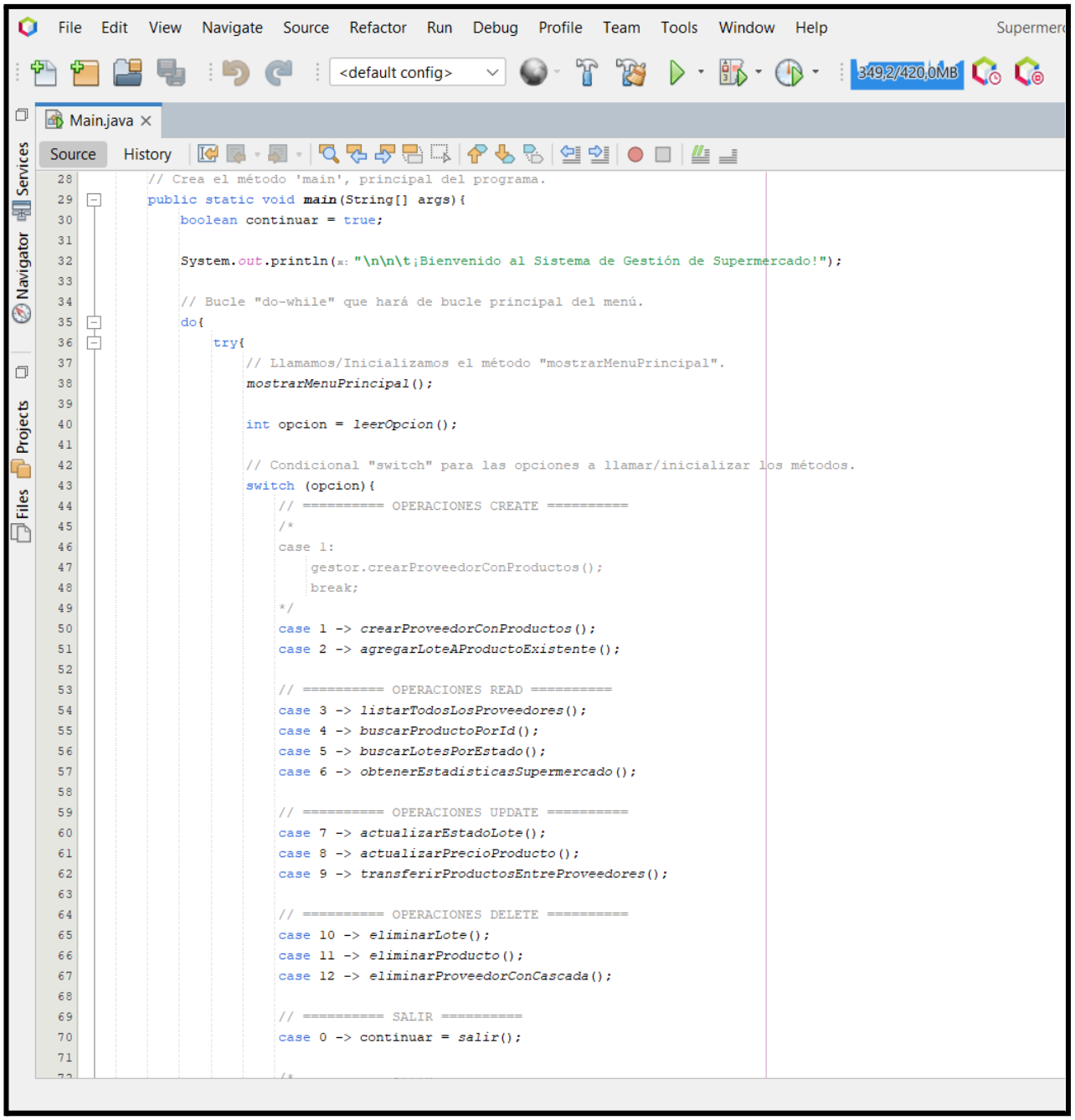
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

---

7. Paso

"Llamada/Inicialización a los distintos métodos u operaciones del sistema".




```java
// Crea el método 'main', principal del programa.
public static void main(String[] args){
    boolean continuar = true;

    System.out.println(x: "\n\n\t¡Bienvenido al Sistema de Gestión de Supermercado!");

    // Bucle "do-while" que hará de bucle principal del menú.
    do{
        try{
            // Llamamos/Inicializamos el método "mostrarMenuPrincipal".
            mostrarMenuPrincipal();

            int opcion = leerOpcion();

            // Condicional "switch" para las opciones a llamar/inicializar los métodos.
            switch (opcion){
                // ========== OPERACIONES CREATE ==========
                /*
                case 1:
                    gestor.crearProveedorConProductos();
                    break;
                */
                case 1 -> crearProveedorConProductos();
                case 2 -> agregarLoteAProductoExistente();

                // ========== OPERACIONES READ ==========
                case 3 -> listarTodosLosProveedores();
                case 4 -> buscarProductoPorId();
                case 5 -> buscarLotesPorEstado();
                case 6 -> obtenerEstadisticasSupermercado();

                // ========== OPERACIONES UPDATE ==========
                case 7 -> actualizarEstadoLote();
                case 8 -> actualizarPrecioProducto();
                case 9 -> transferirProductosEntreProveedores();

                // ========== OPERACIONES DELETE ==========
                case 10 -> eliminarLote();
                case 11 -> eliminarProducto();
                case 12 -> eliminarProveedorConCascada();

                // ========== SALIR ==========
                case 0 -> continuar = salir();
```

8. Paso

"Explicación del programa".

Para empezar a desarrollar esta última parte del proyecto primero hay que dar pie a la estructura del mismo:

```
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
```

Una vez hecho esto, podremos entender cómo es el funcionamiento de nuestra aplicación. Empezando por el "pom.xml" que es el archivo de configuración de Maven donde definimos todas las dependencias necesarias para el proyecto. En este archivo hemos incluido tres dependencias:

- "MySQL Connector - J" para la conexión con la base de datos.
- "Hibernate Core" para el mapeo objeto-relacional.
- "Javax Persistence API (JPA)" que proporciona las anotaciones JPA que utilizaremos en nuestras entidades.

Por otro lado, Maven se encargará automáticamente de descargar estas librerías y gestionar sus dependencias transitivas, permitiéndonos centrarnos en el desarrollo sin preocuparnos por la gestión manual de las librerías.

Siguiendo con el desarrollo de la app tenemos “hibernate.cfg.xml” contiene toda la configuración de Hibernate. Aquí especificamos los parámetros de conexión a la base de datos MySQL (URL, usuario, contraseña, etc.). También definimos el dialecto de MySQL (dialecto SQL) que Hibernate utilizará para generar las consultas SQL optimizadas para este motor de base de datos. Además, hemos activado propiedades de depuración como “show_sql” y “format_sql” que nos permiten visualizar en la consola todas las sentencias SQL que Hibernate genera automáticamente, facilitando el seguimiento y la comprensión de las operaciones realizadas. Posteriormente definiremos “hbm2ddl.auto” configurándola en “validate”, con ello conseguiremos validar/permitir la gestión automática del esquema de la base de datos cuando se detecten cambios en las entidades. Finalmente, en este archivo mapeamos las tres entidades principales del proyecto mediante etiquetas “mapping class", indicando a Hibernate qué clases debe gestionar.

`Por otro lado, la clase “HibernateUtil.java” implementa el patrón de diseño “Singleton” para gestionar el “SessionFactory” de Hibernate. Esta clase contiene un bloque estático, “static”, que se ejecuta al cargar la clase, creando el “SessionFactory” a partir de la configuración definida en “hibernate.cfg.xml”. El método “getSessionFactory()” proporciona acceso a dicha instancia única desde cualquier parte de la aplicación, mientras que el método “shutdown()” se encarga de cerrar correctamente el objeto, el “SessionFactory”, cuando la aplicación finaliza, liberando todos los recursos y conexiones con la base de datos. En definitiva, esto permite garantizar un uso eficiente de los recursos y evitar problemas de concurrencia.

Siguiendo con la estructura del proyecto, encontramos el paquete “modelo”. Dentro de dicho paquete se hallan tres clases java, cada una representando a una entidad del programa/sistema:

- “Proveedor.java” es una de las tres entidades y representa a los proveedores. Está mapeada a la tabla “proveedores” mediante la anotación “@Entity”. Cada atributo de la clase se mapea a una columna de la tabla utilizando “@Column”, mientras que el campo “id” está marcado con “@Id” como clave primaria y “@GeneratedValue” para indicar que su valor se genera automáticamente. La característica más importante de esta clase es su relación “@OneToMany” con la entidad “Producto”, configurada con “cascade = CascadeType.ALL” y “orphanRemoval = true”, lo que significa e implica que todas y cada una de las operaciones realizadas sobre un proveedor se propagan automáticamente a sus productos, y que los productos sin proveedor asociado se eliminan automáticamente. De igual forma, los métodos “helper” “addProducto()” y “removeProducto()” son fundamentales para mantener la consistencia bidireccional de esta relación, actualizando tanto la lista de productos en el proveedor como la referencia del proveedor en cada producto.

- "Producto.java" es la segunda de las tres entidades y representa a los productos provistos. Es una entidad clave para el proyecto, pues es la posición central del modelo, estableciendo relaciones para ambas direcciones, tanto para "Proveedor" con "@ManyToOne" (para cargar el proveedor sólo cuando sea necesario, optimizando así el rendimiento) como para "Lote" con "@OneToMany" (también configurada con cascada completa para conseguir esa bidireccionalidad y concurrencia de datos). Por otro lado, su mapeo se rige a la tabla "productos" y tiene una estructura de anotaciones similar a la de "Proveedor", con la diferencia de campos presentes como "códigoBarras" que está marcada como "unique = true", garantizando que no puedan existir dos productos con el mismo código de barras en la base de datos. Esta clase también implementa métodos "helper" "addLote()" y "removeLote()" que mantienen la coherencia entre la lista de lotes del producto y la referencia del producto con/para cada lote al que corresponde.

- "Lote.java" es la tercera y última de las tres entidades y representa a los lotes, existencias físicas e iguales entre sí de los productos registrados, originales, a la venta por el proveedor. A diferencia de las anteriores entidades, ésta contiene un método "enum", una enumeración interna con la que representar y designar a cada lote uno de los estados que define los estados posibles dentro del supermercado, "EstadoLote" (DISPONIBLE, VENDIDO, CADUCADO y RETIRADO). Éste método al igual que en el resto de entidades está mapeado mediante "@Enumerated(EnumType.STRING)", lo que hace que Hibernate almacene el nombre del "enum" como cadena de texto en lugar de su posición ordinal, proporcionando mayor legibilidad y evitando problemas en caso de que se reordene o altere su estado/valor el "enum" en el futuro. Es una entidad con relación "@ManyToOne" con "Producto" utiliza también carga perezosa "LAZY", lo que significa que el producto asociado sólo se cargará de la base de datos cuando se acceda explícitamente al método "getProducto()".

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero **Módulo:** Acceso a datos (AED)

La clase “GestionSupermercado.java” contiene toda la lógica funcional del programa, todos los métodos/operaciones CRUD del sistema. Cada método maneja su propia transacción siguiendo el patrón “try-with-resources” para garantizar que las sesiones de Hibernate se cierren automáticamente, incluso si ocurre una excepción/error durante la ejecución. Los métodos de creación “Operaciones Create” demuestran el poder de las operaciones en cascada: al persistir un proveedor con “session.persist(proveedor)”, Hibernate se encarga automáticamente de guardar todos sus productos y los lotes de cada producto, reduciendo significativamente la cantidad de código necesario, ergo libera carga significativa al código. Los métodos de consulta “Operaciones Read” utilizan sentencias HQL (Hibernate Query Language), un lenguaje orientado a objetos similar a SQL pero que trabaja con clases y atributos en lugar de tablas y columnas, es decir, trabaja directamente con objetos y no con entidades de tablas y columnas. Por ejemplo, la consulta “FROM Lote l WHERE l.estado = :estado” recupera lotes filtrando por su estado, e Hibernate la traduce automáticamente al SQL apropiado para MySQL. Los métodos de actualización “Operaciones Update” emplean “session.merge()” para entidades existentes, e Hibernate aplica su mecanismo de "dirty checking" para detectar/saber qué campos han cambiado y así poder generar sentencias “UPDATE” que solo modifican dichos campos específicos. Los métodos de eliminación “Operaciones Delete” demuestran el funcionamiento completo de las cascadas. Es decir, eliminar un proveedor provoca la eliminación automática de todos sus productos, y estos a su vez eliminan todos sus lotes, todo ello en una sola operación que mantiene la integridad referencial de la base de datos.

La clase “Main.java” proporciona la pieza restante del código, una interfaz de usuario mediante un menú interactivo por consola para una interacción constante con el usuario. Este menú está organizado en cuatro secciones claramente diferenciadas: ‘operaciones CREATE’ (opciones 1-2), ‘operaciones READ’ (3-6), ‘operaciones UPDATE’ (7-9) y ‘operaciones DELETE’ (10-12), además de una última opción con la que poder salir del programa (0). La aplicación comienza por ejecutarse en un bucle “do-while” que solo termina cuando el usuario selecciona la opción de salida, (0). La gestión de errores es robusta, capturando “InputMismatchException” cuando el usuario introduce valores no esperados o incorrectos, como lo son los no numéricos y limpiando el buffer del escáner “user” para evitar bucles infinitos. Las operaciones destructivas “DELETE” solicitan confirmación explícita del usuario antes de proceder, argumentando o explicando lo que conllevaría llevar a cabo la acción de ejecutar dicha operación, mostrando advertencias sobre las consecuencias de la cascada. Y para finalizar, el método “salir()” invoca “HibernateUtil.shutdown()” para cerrar correctamente el “SessionFactory” y liberar todos los recursos, garantizando una terminación limpia de la aplicación. Este diseño ofrece una experiencia de usuario clara y segura, permitiendo probar todas las funcionalidades del sistema de forma intuitiva, clara y sencilla.

Por otro lado, a lo largo del código (de todos los archivos) encontrará comentarios claros, limpios, intuitivos de entender e indicativos sobre qué está pasando en ese momento, qué ocurre, para qué es la línea de código, que función cumple el método o la clase, etc.

En cuanto a las dificultades encontradas durante el desarrollo y trabajo de esta práctica me gustaría resaltar un par de cosas. Empezando por entender cómo sería el funcionamiento del programa dada la indicación explícita en la tarea de que hubiese bidireccionalidad entre tablas. Este conflicto se resolvió en clase tras la explicación del profesor, de Javier.

Otra cuestión fue la implementación de nuevos métodos como los "helper" con los que mantener un programa bidireccional y con concurrencia de datos. Para ello lo que hice fue ir trasteando en un proyecto independiente/ajeno a este con la ayuda de dos AI's (ChatGPT y Claude) para tener dos puntos de referencia y no basarme en la primera respuesta que me diese solo una de ellas.

Si bien es cierto que usé ambas AI's a lo largo de todo el proyecto, mi forma de trabajar era la siguiente, primero leía las instrucciones y peticiones de la tarea e intentaba hacerlo yo, después mandaba el "prompt" de la parte que estaba trabajando (no de todo el archivo o proyecto) y comparaba las dos respuestas dadas y lo combinaba con lo que ya tenía hecho/avanzado. También me he apoyado bastante en esta/s herramienta/s para los comentarios, pidiéndoles que me explicasen una parte concreta del código, una vez leída y entendida la explicación, trasladaba de forma más resumida, propia y escueta lo que yo había entendido de la misma.

Y finalmente, también me gustaría hacer hincapié en el uso de HIbernate teniendo en cuenta que es un procedimiento relativamente nuevo para nosotros. Desde mi punto de vista, el uso de este sistema no solo me ha permitido trabajar más fácilmente, sino que me ha sido una metodología mucho más fácil y rápida de comprender y trabajar a diferencia del método que aplicábamos antes que era únicamente realizar la correspondencia objeto-relacional mediante archivos "XML".