

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](https://github.com/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.
-

Alumno: Rodrigo López Pérez

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)

ÍNDICE

1. Apartado 1. - Preparación del entorno.....	3
1.1. Apartado 1.1. - Creación de la base de datos.....	3
1.2. Apartado 1.2. - Configuración del proyecto Maven.....	4
1.3. Apartado 1.3. - Configuración de Hibernate.....	7
1.4. Apartado 1.4. - Clase HibernateUtil (paquete com.dam.util).....	8
2. Apartado 2. - Implementación de entidades.....	9
3. Apartado 3. - Operaciones Create.....	12
4. Apartado 4. - Operaciones Read.....	14
5. Apartado 5. - Operaciones Update.....	18
6. Apartado 6. - Operaciones Delete.....	21
7. Apartado 7. - Clase principal y menú.....	24
8. Apartado 8. - Explicación del programa.....	25

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”.

The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'Schemas' tree with 'supermercado_hibernate' selected. The main window shows a SQL script for creating the database and inserting data. The 'Result Grid' displays the output of the queries, showing the creation of the database and the insertion of data into the 'lotes' table.

```
-- Script de creación de base de datos para el sistema de gestión de supermercado
-- Autor: Javier Sebastián Herrero
-- Fecha: Diciembre 2025

-- Eliminar la base de datos si existe y crearla nuevamente
6 DROP DATABASE IF EXISTS supermercado_hibernate;
7 CREATE DATABASE supermercado_hibernate CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
8 USE supermercado_hibernate;
```

proveedor	producto	precio	codigo_lote	cantidad	fecha_caducidad	estado
Alimentaria Central S.A.	Atún en aceite 3 latas	4.50	LOT-ATUN-001	80	2027-06-30	DISPONIBLE
Alimentaria Central S.A.	Atún en aceite 3 latas	4.50	LOT-ATUN-002	0	2027-08-15	VENDIDO
Alimentaria Central S.A.	Pasta italiana 500g	1.25	LOT-PAST-001	150	2028-03-15	DISPONIBLE
Alimentaria Central S.A.	Pasta italiana 500g	1.25	LOT-PAST-002	100	2028-04-20	DISPONIBLE
Alimentaria Central S.A.	Tomate triturado 400g	0.89	LOT-TOM-001	200	2025-12-31	DISPONIBLE
Alimentaria Central S.A.	Tomate triturado 400g	0.89	LOT-TOM-002	50	2025-10-15	CADUCADO
Frutas y Verduras del Campo	Manzanas Golden 3kg	2.35	LOT-MANZ-001	30	2025-01-10	DISPONIBLE
Frutas y Verduras del Campo	Paltanos de Canarias 3kg	1.95	LOT-PLAT-001	45	2025-01-05	DISPONIBLE
Frutas y Verduras del Campo	Tomates cherry 250g	1.65	LOT-CHE-001	25	2025-01-08	DISPONIBLE
Lácteos La Granja	Leche entera 1L	1.10	LOT-LECH-001	120	2025-01-20	DISPONIBLE
Lácteos La Granja	Leche entera 1L	1.10	LOT-LECH-002	80	2025-01-25	DISPONIBLE

Output:

#	Time	Action	Message	Duration / Fetch
19	16:50:41	INSERT INTO lotes (codigo, cantidad, fecha_caducidad, estado, id_producto) VALUES (LOT-MANZ-001, 30, ...)	1 row(s) affected	0.016 sec
20	16:50:41	INSERT INTO lotes (codigo, cantidad, fecha_caducidad, estado, id_producto) VALUES (LOT-PLAT-001, 45, ...)	1 row(s) affected	0.015 sec
21	16:50:41	INSERT INTO lotes (codigo, cantidad, fecha_caducidad, estado, id_producto) VALUES (LOT-CHE-001, 25, ...)	1 row(s) affected	0.000 sec
22	16:50:41	INSERT INTO lotes (codigo, cantidad, fecha_caducidad, estado, id_producto) VALUES (LOT-LECH-001, 120, ...)	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0	0.000 sec
23	16:50:41	INSERT INTO lotes (codigo, cantidad, fecha_caducidad, estado, id_producto) VALUES (LOT-YOG-001, 60, ...)	1 row(s) affected	0.000 sec
24	16:50:41	INSERT INTO lotes (codigo, cantidad, fecha_caducidad, estado, id_producto) VALUES (LOT-GUIES-001, 15, ...)	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0	0.016 sec
25	16:50:41	SELECT p.nombre AS proveedor, pr.nombre AS producto, pr.precio, l.codigo AS codigo_lote, l.c... 14 row(s) returned		0.000 sec / 0.000 sec

Alumno: Rodrigo López Pérez

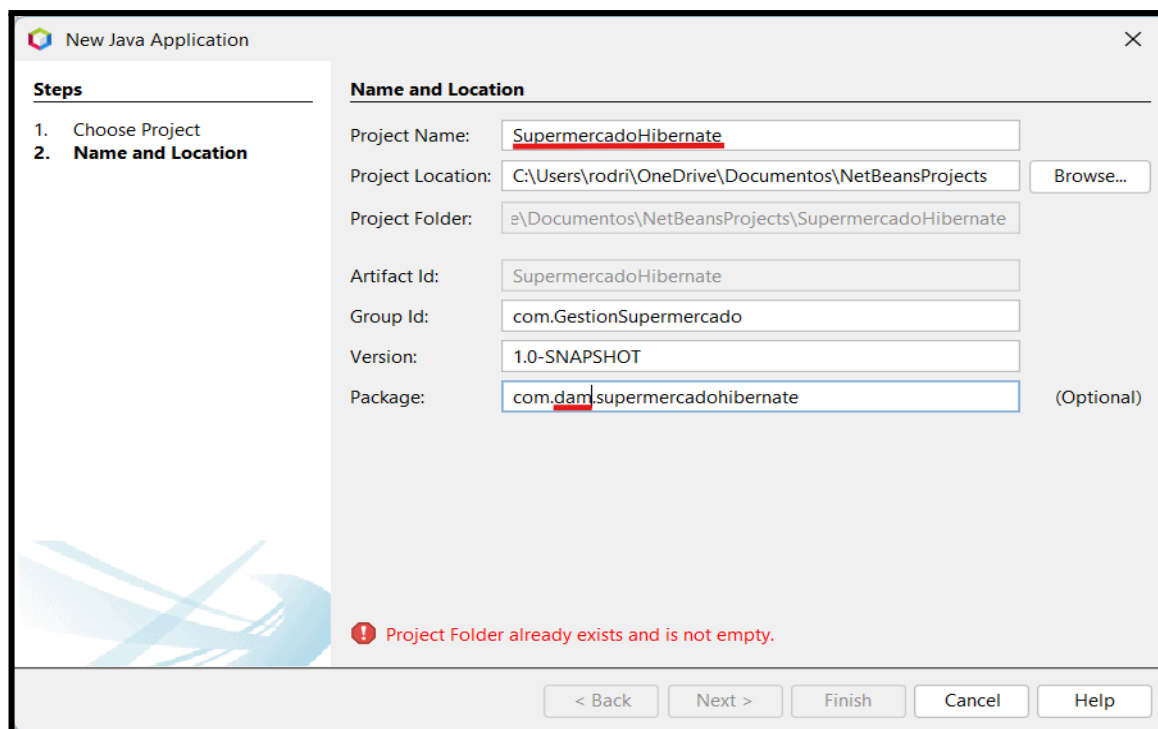
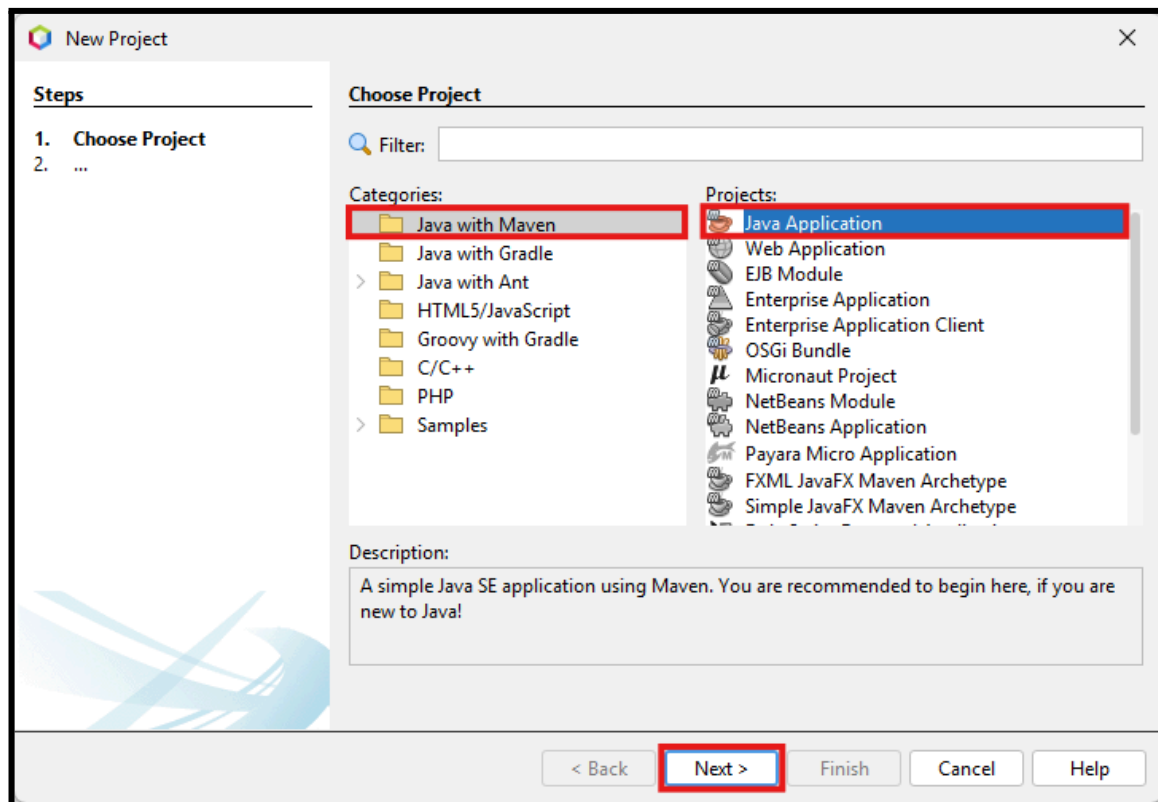
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)

1.2. Paso

“Configuración del proyecto Maven”.

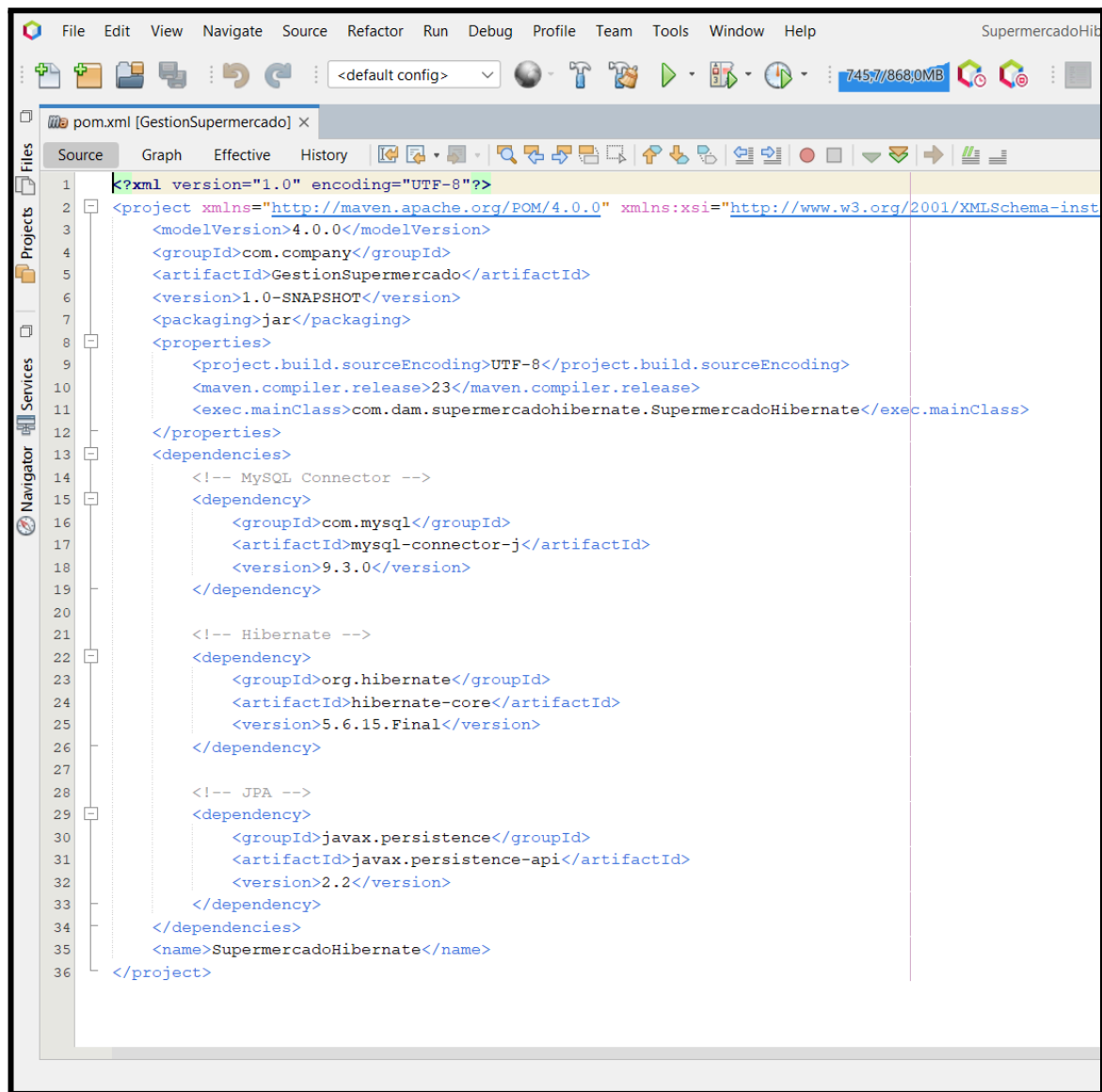


Alumno: Rodrigo López Pérez

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)

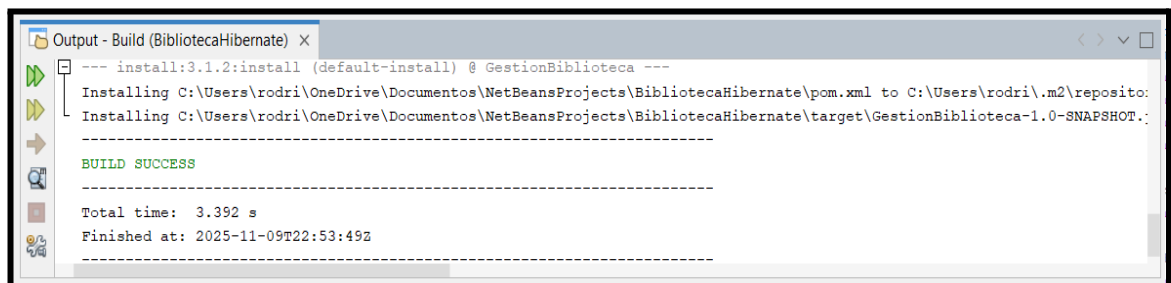


The screenshot shows the NetBeans IDE interface with the 'pom.xml' file open. The file contains the following XML content:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst
<modelVersion>4.0.0</modelVersion>
<groupId>com.company</groupId>
<artifactId>GestionSupermercado</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.release>23</maven.compiler.release>
  <exec.mainClass>com.dam.supermercadohibernate.SupermercadoHibernate</exec.mainClass>
</properties>
<dependencies>
  <!-- MySQL Connector -->
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>9.3.0</version>
  </dependency>

  <!-- Hibernate -->
  <dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.6.15.Final</version>
  </dependency>

  <!-- JPA -->
  <dependency>
    <groupId>javax.persistence</groupId>
    <artifactId>javax.persistence-api</artifactId>
    <version>2.2</version>
  </dependency>
</dependencies>
<name>SupermercadoHibernate</name>
</project>
```



The screenshot shows the 'Output - Build (BibliotecaHibernate)' window. The output text is as follows:

```
--- install:3.1.2:install (default-install) @ GestionBiblioteca ---
Installing C:\Users\rodri\OneDrive\Documentos\NetBeansProjects\BibliotecaHibernate\pom.xml to C:\Users\rodri\.m2\reposito:
Installing C:\Users\rodri\OneDrive\Documentos\NetBeansProjects\BibliotecaHibernate\target\GestionBiblioteca-1.0-SNAPSHOT.:
-----
BUILD SUCCESS
-----
Total time: 3.392 s
Finished at: 2025-11-09T22:53:49Z
-----
```

Alumno: Rodrigo López Pérez

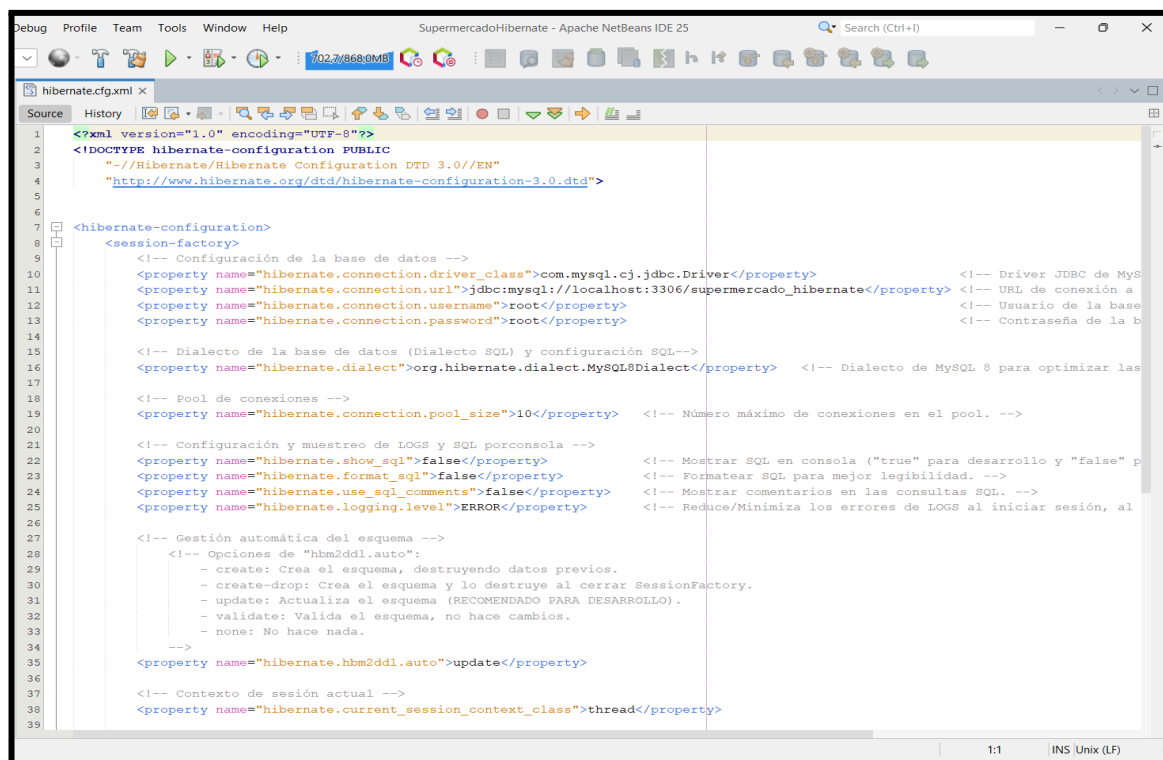
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

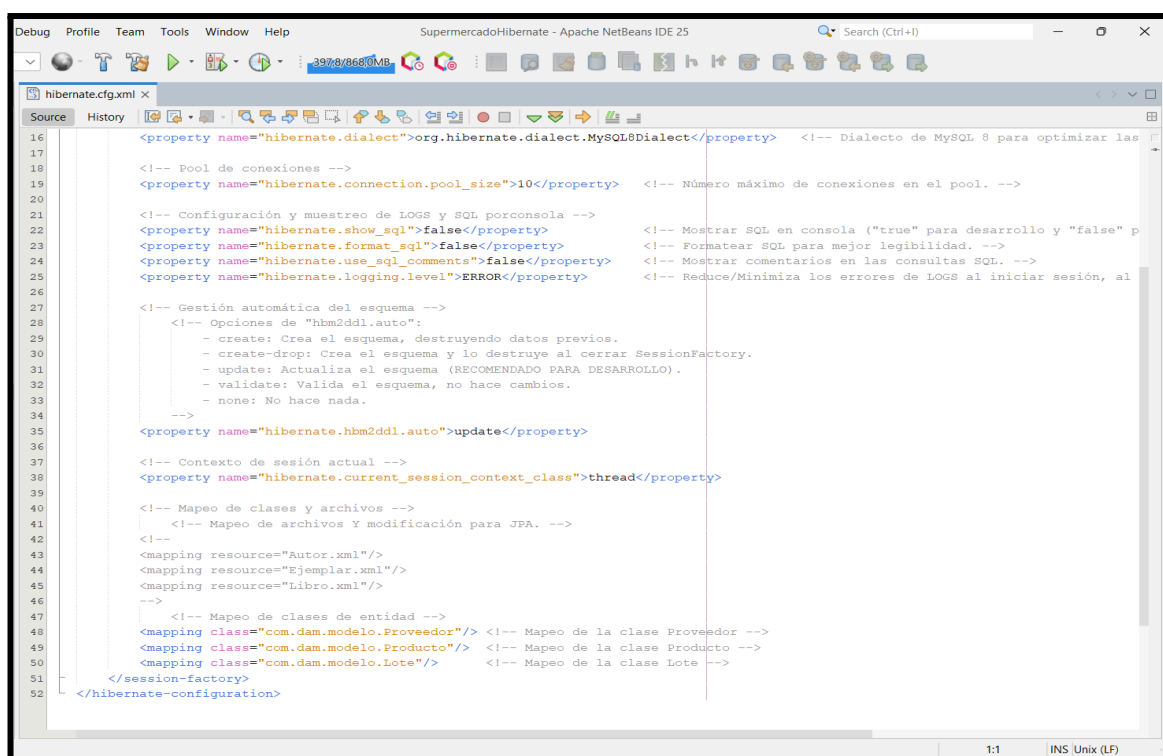
Módulo: Acceso a datos (AED)

1.3. Paso

“Configuración de Hibernate”.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE hibernate-configuration PUBLIC
3     "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4     "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
5
6
7 <hibernate-configuration>
8     <session-factory>
9         <!-- Configuración de la base de datos -->
10        <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property> <!-- Driver JDBC de Mys
11        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/supermercado_hibernate</property> <!-- URL de conexión a
12        <property name="hibernate.connection.username">root</property> <!-- Usuario de la base
13        <property name="hibernate.connection.password">root</property> <!-- Contraseña de la b
14
15        <!-- Dialecto de la base de datos (Dialecto SQL) y configuración SQL-->
16        <property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property> <!-- Dialecto de MySQL 8 para optimizar las
17
18        <!-- Pool de conexiones -->
19        <property name="hibernate.connection.pool_size">10</property> <!-- Número máximo de conexiones en el pool. -->
20
21        <!-- Configuración y muestreo de LOGS y SQL porconsola -->
22        <property name="hibernate.show_sql">>false</property> <!-- Mostrar SQL en consola ("true" para desarrollo y "false" p
23        <property name="hibernate.format_sql">>false</property> <!-- Formatear SQL para mejor legibilidad. -->
24        <property name="hibernate.use_sql_comments">>false</property> <!-- Mostrar comentarios en las consultas SQL. -->
25        <property name="hibernate.logging.level">ERROR</property> <!-- Reduce/Minimiza los errores de LOGS al iniciar sesión, al
26
27        <!-- Gestión automática del esquema -->
28        <!-- Opciones de "hbm2ddl.auto":
29        - create: Crea el esquema, destruyendo datos previos.
30        - create-drop: Crea el esquema y lo destruye al cerrar SessionFactory.
31        - update: Actualiza el esquema (RECOMENDADO PARA DESARROLLO).
32        - validate: Valida el esquema, no hace cambios.
33        - none: No hace nada.
34        -->
35        <property name="hibernate.hbm2ddl.auto">update</property>
36
37        <!-- Contexto de sesión actual -->
38        <property name="hibernate.current_session_context_class">thread</property>
39    </session-factory>
40</hibernate-configuration>
```



```
16 <property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property> <!-- Dialecto de MySQL 8 para optimizar las
17
18 <!-- Pool de conexiones -->
19 <property name="hibernate.connection.pool_size">10</property> <!-- Número máximo de conexiones en el pool. -->
20
21 <!-- Configuración y muestreo de LOGS y SQL porconsola -->
22 <property name="hibernate.show_sql">>false</property> <!-- Mostrar SQL en consola ("true" para desarrollo y "false" p
23 <property name="hibernate.format_sql">>false</property> <!-- Formatear SQL para mejor legibilidad. -->
24 <property name="hibernate.use_sql_comments">>false</property> <!-- Mostrar comentarios en las consultas SQL. -->
25 <property name="hibernate.logging.level">ERROR</property> <!-- Reduce/Minimiza los errores de LOGS al iniciar sesión, al
26
27 <!-- Gestión automática del esquema -->
28 <!-- Opciones de "hbm2ddl.auto":
29 - create: Crea el esquema, destruyendo datos previos.
30 - create-drop: Crea el esquema y lo destruye al cerrar SessionFactory.
31 - update: Actualiza el esquema (RECOMENDADO PARA DESARROLLO).
32 - validate: Valida el esquema, no hace cambios.
33 - none: No hace nada.
34 -->
35 <property name="hibernate.hbm2ddl.auto">update</property>
36
37 <!-- Contexto de sesión actual -->
38 <property name="hibernate.current_session_context_class">thread</property>
39
40 <!-- Mapeo de clases y archivos -->
41 <!-- Mapeo de archivos Y modificación para JPA. -->
42 <!--
43 <mapping resource="Autor.xml"/>
44 <mapping resource="Ejemplar.xml"/>
45 <mapping resource="Libro.xml"/>
46 -->
47 <!-- Mapeo de clases de entidad -->
48 <mapping class="com.dam.modelo.Proveedor"/> <!-- Mapeo de la clase Proveedor -->
49 <mapping class="com.dam.modelo.Producto"/> <!-- Mapeo de la clase Producto -->
50 <mapping class="com.dam.modelo.Lote"/> <!-- Mapeo de la clase Lote -->
51 </session-factory>
52 </hibernate-configuration>
```

Alumno: Rodrigo López Pérez

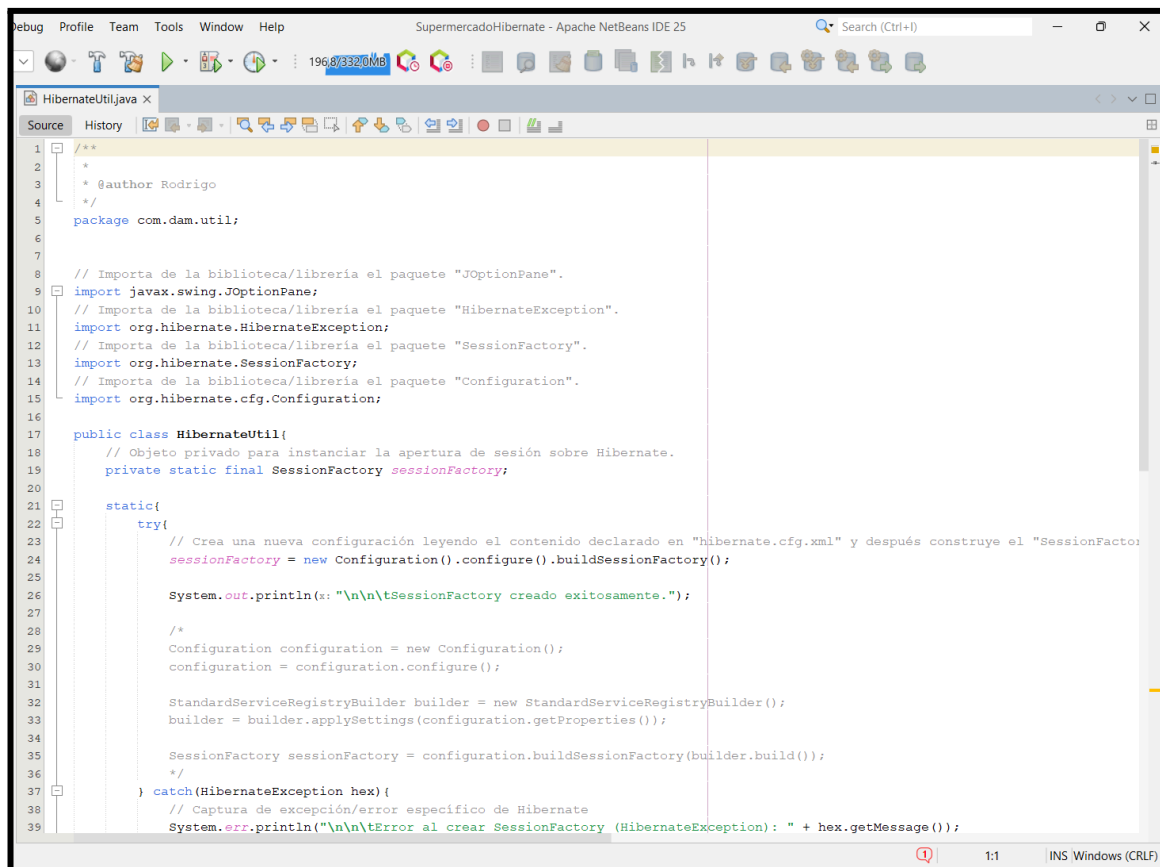
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

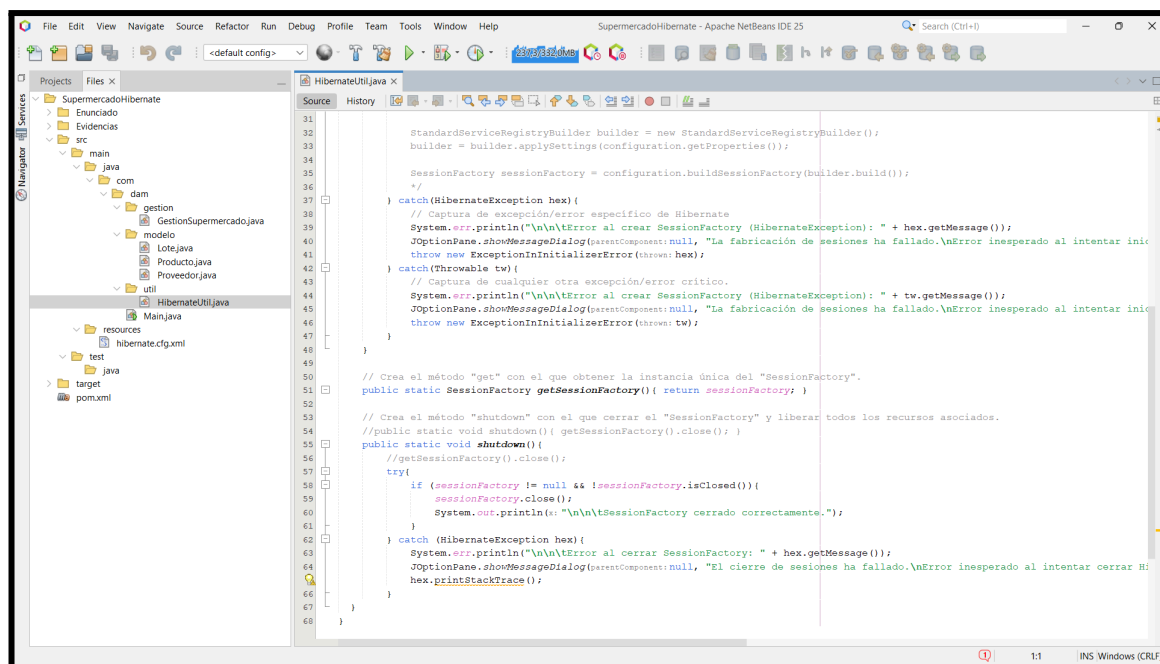
Módulo: Acceso a datos (AED)

1.4. Paso

“Clase HibernateUtil (paquete com.dam.util)”.



```
1  /**
2   *
3   * @author Rodrigo
4   */
5   package com.dam.util;
6
7
8   // Importa de la biblioteca/librería el paquete "JOptionPane".
9   import javax.swing.JOptionPane;
10  // Importa de la biblioteca/librería el paquete "HibernateException".
11  import org.hibernate.HibernateException;
12  // Importa de la biblioteca/librería el paquete "SessionFactory".
13  import org.hibernate.SessionFactory;
14  // Importa de la biblioteca/librería el paquete "Configuration".
15  import org.hibernate.cfg.Configuration;
16
17  public class HibernateUtil{
18      // Objeto privado para instanciar la apertura de sesión sobre Hibernate.
19      private static final SessionFactory sessionFactory;
20
21
22      static{
23          try{
24              // Crea una nueva configuración leyendo el contenido declarado en "hibernate.cfg.xml" y después construye el "SessionFactory".
25              sessionFactory = new Configuration().configure().buildSessionFactory();
26
27              System.out.println("Se creó el SessionFactory correctamente.");
28
29              /*
30               Configuration configuration = new Configuration();
31               configuration = configuration.configure();
32
33               StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder();
34               builder = builder.applySettings(configuration.getProperties());
35
36               SessionFactory sessionFactory = configuration.buildSessionFactory(builder.build());
37               */
38          } catch (HibernateException hex) {
39              // Captura de excepción/error específico de Hibernate
40              System.err.println("Error al crear SessionFactory (HibernateException): " + hex.getMessage());
41          }
42      }
43  }
```



```
43
44
45      // Captura de cualquier otra excepción/error crítico.
46      System.err.println("Error al crear SessionFactory (HibernateException): " + tw.getMessage());
47      JOptionPane.showMessageDialog(null, "La fabricación de sesiones ha fallado.\nError inesperado al intentar iniciar la sesión.");
48      throw new ExceptionInInitializerError(thrown: tw);
49  }
50
51  // Crea el método "get" con el que obtener la instancia única del "SessionFactory".
52  public static SessionFactory getSessionFactory() { return sessionFactory; }
53
54  // Crea el método "shutdown" con el que cerrar el "SessionFactory" y liberar todos los recursos asociados.
55  public static void shutdown() { getSessionFactory().close(); }
56
57  // getSessionFactory().close();
58  try{
59      if (sessionFactory != null && !sessionFactory.isClosed()){
60          sessionFactory.close();
61          System.out.println("Se cerró el SessionFactory correctamente.");
62      }
63  } catch (HibernateException hex) {
64      System.err.println("Error al cerrar SessionFactory: " + hex.getMessage());
65      JOptionPane.showMessageDialog(null, "El cierre de sesiones ha fallado.\nError inesperado al intentar cerrar la sesión.");
66      hex.printStackTrace();
67  }
68  }
```

Alumno: Rodrigo López Pérez

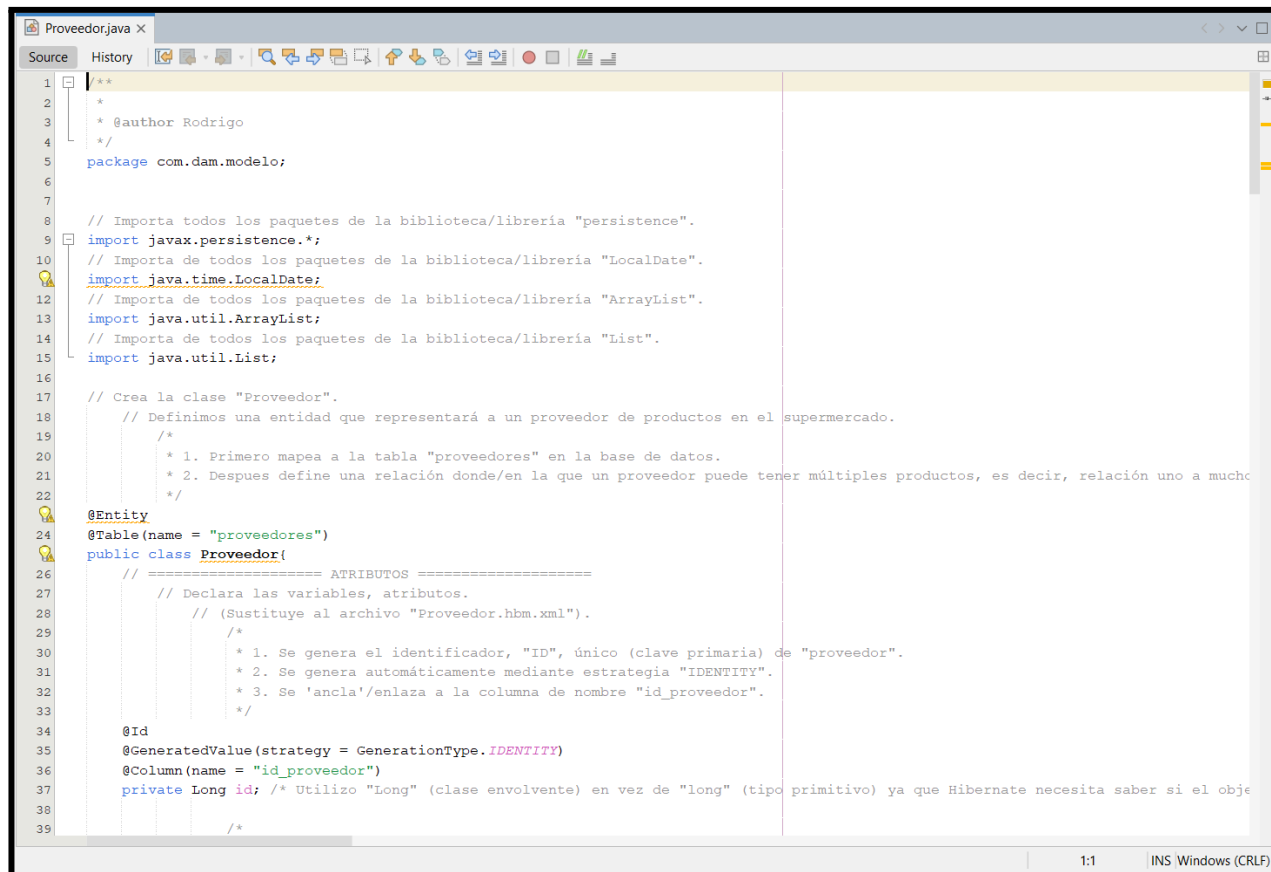
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)

2. Paso

“Implementación de entidades”.



```
1  /**
2   *
3   * @author Rodrigo
4   */
5  package com.dam.modelo;
6
7
8  // Importa todos los paquetes de la biblioteca/librería "persistence".
9  import javax.persistence.*;
10 // Importa de todos los paquetes de la biblioteca/librería "LocalDate".
11 import java.time.LocalDate;
12 // Importa de todos los paquetes de la biblioteca/librería "ArrayList".
13 import java.util.ArrayList;
14 // Importa de todos los paquetes de la biblioteca/librería "List".
15 import java.util.List;
16
17 // Crea la clase "Proveedor".
18 // Definimos una entidad que representará a un proveedor de productos en el supermercado.
19 /**
20  * 1. Primero mapea a la tabla "proveedores" en la base de datos.
21  * 2. Después define una relación donde/en la que un proveedor puede tener múltiples productos, es decir, relación uno a muchos.
22  */
23 @Entity
24 @Table(name = "proveedores")
25 public class Proveedor {
26     // ===== ATRIBUTOS =====
27     // Declara las variables, atributos.
28     // (Sustituye al archivo "Proveedor.hbm.xml").
29     /**
30      * 1. Se genera el identificador, "ID", único (clave primaria) de "proveedor".
31      * 2. Se genera automáticamente mediante estrategia "IDENTITY".
32      * 3. Se 'ancla'/enlaza a la columna de nombre "id_proveedor".
33      */
34     @Id
35     @GeneratedValue(strategy = GenerationType.IDENTITY)
36     @Column(name = "id_proveedor")
37     private Long id; /* Utilizo "Long" (clase envolvente) en vez de "long" (tipo primitivo) ya que Hibernate necesita saber si el objeto es primitivo o envuelto. */
38
39     /**
```

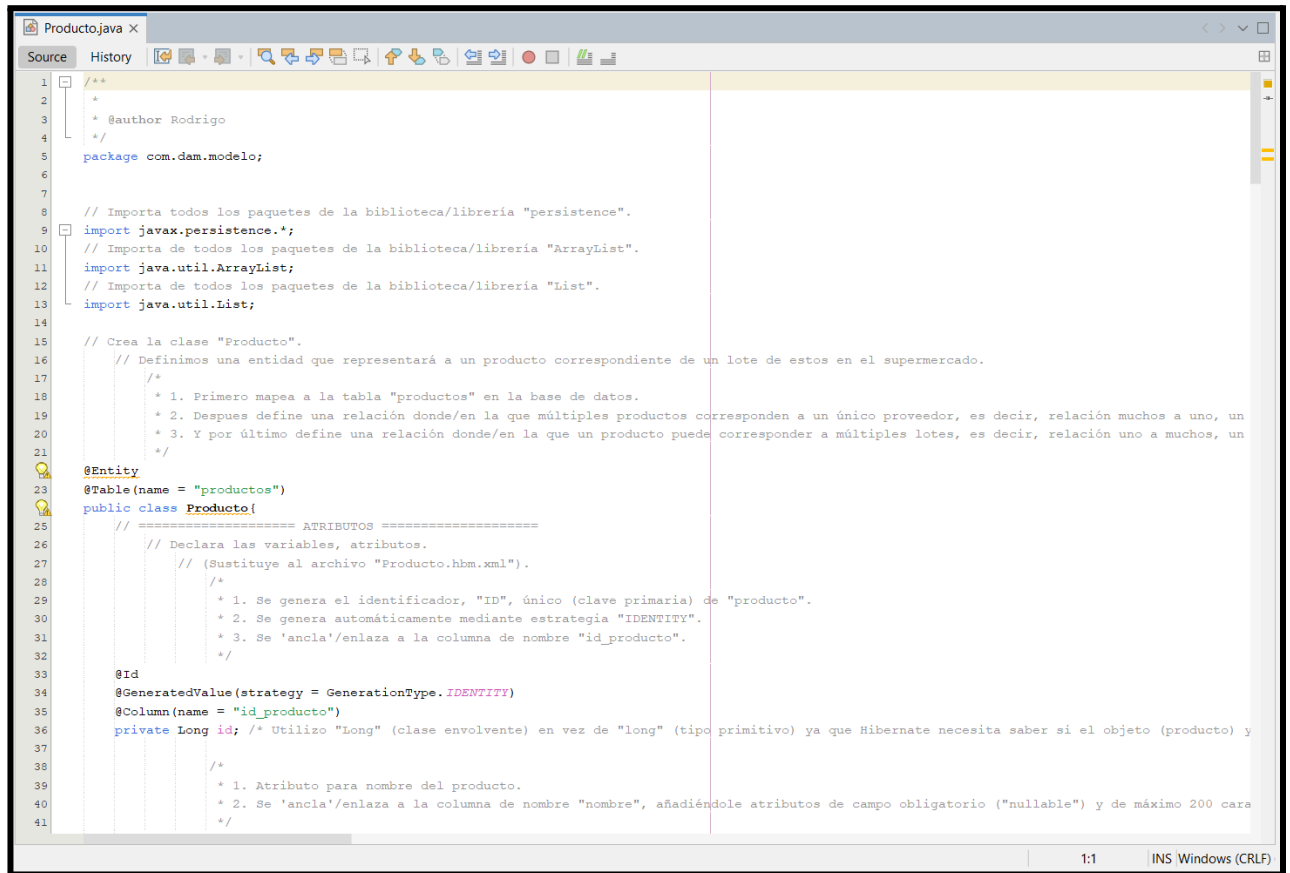
Entidad 'Proveedor' (Proveedor.java).

Alumno: Rodrigo López Pérez

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)



```
1  /**
2   *
3   * @author Rodrigo
4   */
5   package com.dam.modelo;
6
7
8   // Importa todos los paquetes de la biblioteca/librería "persistence".
9   import javax.persistence.*;
10  // Importa de todos los paquetes de la biblioteca/librería "ArrayList".
11  import java.util.ArrayList;
12  // Importa de todos los paquetes de la biblioteca/librería "List".
13  import java.util.List;
14
15  // Crea la clase "Producto".
16  // Definimos una entidad que representará a un producto correspondiente de un lote de estos en el supermercado.
17  /**
18   * 1. Primero mapea a la tabla "productos" en la base de datos.
19   * 2. Después define una relación donde/en la que múltiples productos corresponden a un único proveedor, es decir, relación muchos a uno, un
20   * 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
21   */
22  @Entity
23  @Table(name = "productos")
24  public class Producto{
25      // ===== ATRIBUTOS =====
26      // Declara las variables, atributos.
27      // (Sustituye al archivo "Producto.hbm.xml").
28      /**
29       * 1. Se genera el identificador, "ID", único (clave primaria) de "producto".
30       * 2. Se genera automáticamente mediante estrategia "IDENTITY".
31       * 3. Se 'ancla'/enlaza a la columna de nombre "id_producto".
32       */
33      @Id
34      @GeneratedValue(strategy = GenerationType.IDENTITY)
35      @Column(name = "id_producto")
36      private Long id; /* Utilizo "Long" (clase envolvente) en vez de "long" (tipo primitivo) ya que Hibernate necesita saber si el objeto (producto) y
37
38      /**
39       * 1. Atributo para nombre del producto.
40       * 2. Se 'ancla'/enlaza a la columna de nombre "nombre", añadiéndole atributos de campo obligatorio ("nullable") y de máximo 200 cara
41      */
```

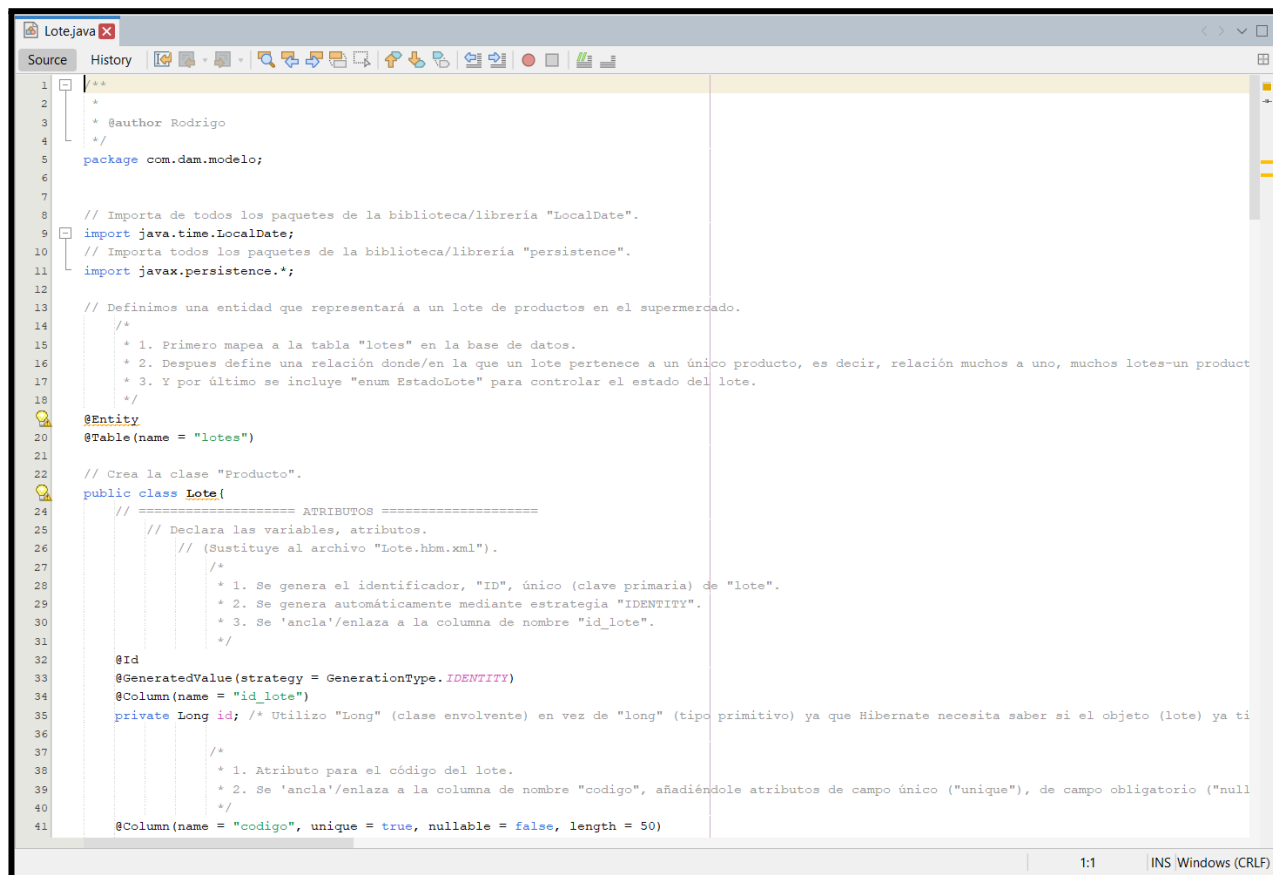
Entidad 'Producto' (Producto.java).

Alumno: Rodrigo López Pérez

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)



```
1  /**
2   *
3   * @author Rodrigo
4   */
5   package com.dam.modelo;
6
7
8   // Importa de todos los paquetes de la biblioteca/librería "LocalDate".
9   import java.time.LocalDate;
10  // Importa todos los paquetes de la biblioteca/librería "persistence".
11  import javax.persistence.*;
12
13  // Definimos una entidad que representará a un lote de productos en el supermercado.
14  /**
15   * 1. Primero mapea a la tabla "lotes" en la base de datos.
16   * 2. Después define una relación donde/en la que un lote pertenece a un único producto, es decir, relación muchos a uno, muchos lotes-un producto.
17   * 3. Y por último se incluye "enum EstadoLote" para controlar el estado del lote.
18   */
19  @Entity
20  @Table(name = "lotes")
21
22  // Crea la clase "Producto".
23  public class Lote{
24      // ===== ATRIBUTOS =====
25      // Declara las variables, atributos.
26      // (Sustituye al archivo "Lote.hbm.xml").
27      /**
28       * 1. Se genera el identificador, "ID", único (clave primaria) de "lote".
29       * 2. Se genera automáticamente mediante estrategia "IDENTITY".
30       * 3. Se 'ancla'/enlaza a la columna de nombre "id_lote".
31       */
32      @Id
33      @GeneratedValue(strategy = GenerationType.IDENTITY)
34      @Column(name = "id_lote")
35      private Long id; /* Utilizo "Long" (clase envolvente) en vez de "long" (tipo primitivo) ya que Hibernate necesita saber si el objeto (lote) ya tiene un valor. */
36
37      /**
38       * 1. Atributo para el código del lote.
39       * 2. Se 'ancla'/enlaza a la columna de nombre "codigo", añadiéndole atributos de campo único ("unique"), de campo obligatorio ("nullable = false").
40       */
41      @Column(name = "codigo", unique = true, nullable = false, length = 50)
```

Entidad 'Lote' (Lote.java).

Alumno: Rodrigo López Pérez

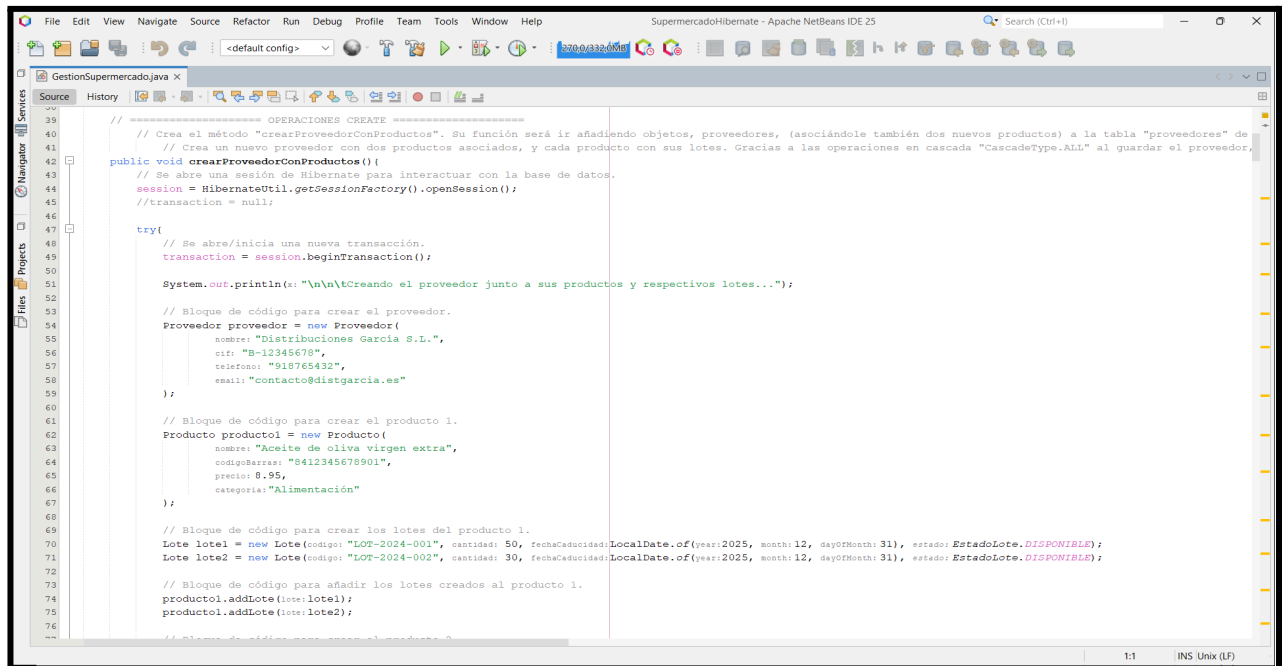
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)

3. Paso

“Operaciones Create”.



```
// ===== OPERACIONES CREATE =====
// Crea el método "crearProveedorConProductos". Su función será ir añadiendo objetos, proveedores, (asociándole también dos nuevos productos) a la tabla "proveedores" de
// Crea un nuevo proveedor con dos productos asociados, y cada producto con sus lotes. Gracias a las operaciones en cascada "CascadeType.ALL" al guardar el proveedor,
public void crearProveedorConProductos() {
    // Se abre una sesión de Hibernate para interactuar con la base de datos.
    session = HibernateUtil.getSessionFactory().openSession();
    //transaction = null;

    try {
        // Se abre/inicia una nueva transacción.
        transaction = session.beginTransaction();

        System.out.println("\n\n\tCreando el proveedor junto a sus productos y respectivos lotes...");

        // Bloque de código para crear el proveedor.
        Proveedor proveedor = new Proveedor(
            nombre: "Distribuciones García S.L.",
            cfr: "B-12345678",
            telefono: "918765432",
            email: "contacto@distgarcia.es"
        );

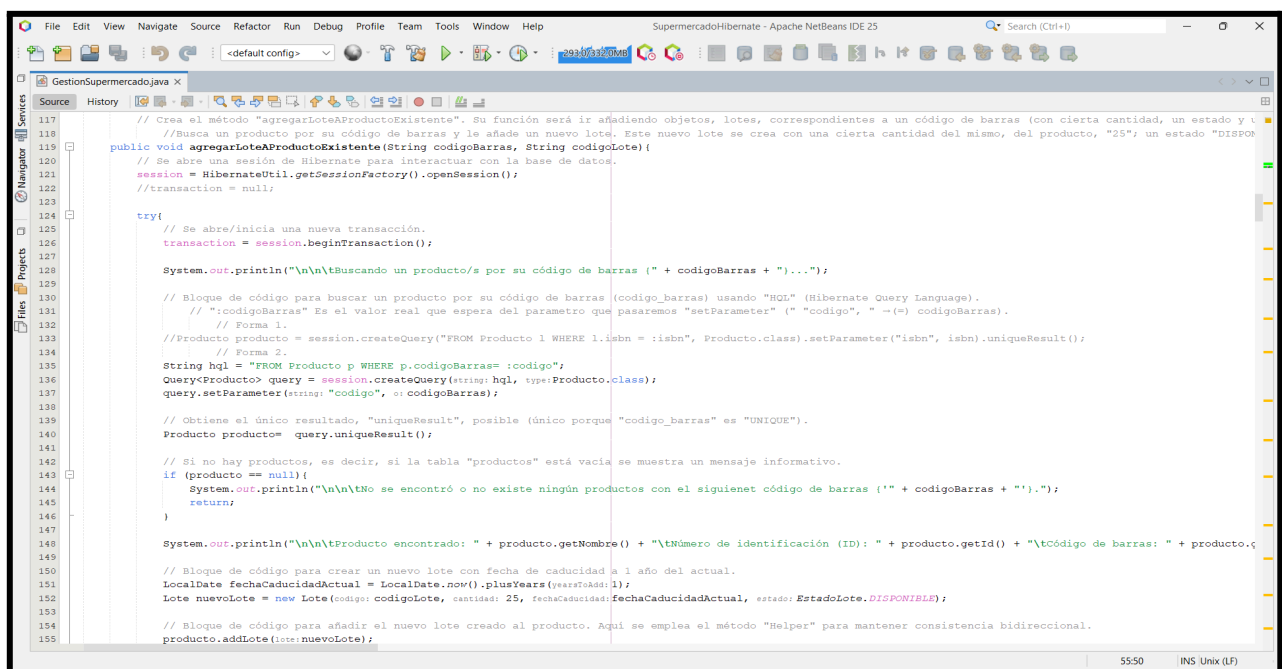
        // Bloque de código para crear el producto 1.
        Producto producto1 = new Producto(
            nombre: "Aceite de oliva virgen extra",
            codigoBarras: "8412345678901",
            precio: 0.95,
            categoria: "Alimentación"
        );

        // Bloque de código para crear los lotes del producto 1.
        Lote lote1 = new Lote(codigo: "LOT-2024-001", cantidad: 50, fechaCaducidad: LocalDate.of(year: 2025, month: 12, dayOfMonth: 31), estado: EstadoLote.DISPONIBLE);
        Lote lote2 = new Lote(codigo: "LOT-2024-002", cantidad: 30, fechaCaducidad: LocalDate.of(year: 2025, month: 12, dayOfMonth: 31), estado: EstadoLote.DISPONIBLE);

        // Bloque de código para añadir los lotes creados al producto 1.
        producto1.addLote(lote1);
        producto1.addLote(lote2);

        // Bloque de código para guardar el producto 1
    }
}
```

Operación ‘crear proveedor/es’ (Método “crearProveedorConProductos()”) (GestionSupermercado.java).



```
// Crea el método "agregarLoteAProductoExistente". Su función será ir añadiendo objetos, lotes, correspondientes a un código de barras (con cierta cantidad, un estado y u
// Busca un producto por su código de barras y le añade un nuevo lote. Este nuevo lote se crea con una cierta cantidad del mismo, del producto, "25"; un estado "DISPON
public void agregarLoteAProductoExistente(String codigoBarras, String codigoLote) {
    // Se abre una sesión de Hibernate para interactuar con la base de datos.
    session = HibernateUtil.getSessionFactory().openSession();
    //transaction = null;

    try {
        // Se abre/inicia una nueva transacción.
        transaction = session.beginTransaction();

        System.out.println("\n\n\tBuscando un producto/s por su código de barras (" + codigoBarras + ")...");

        // Bloque de código para buscar un producto por su código de barras (codigo_barras) usando "HQL" (Hibernate Query Language).
        // "codigoBarras" Es el valor real que espera del parametro que pasaremos "setParameter" ("codigo", "=> codigoBarras).
        // Forma 1.
        // Producto producto = session.createQuery("FROM Producto p WHERE p.codigoBarras = :codigo", Producto.class).setParameter("codigo", codigoBarras).uniqueResult();
        // Forma 2.
        String hql = "FROM Producto p WHERE p.codigoBarras = :codigo";
        Query<Producto> query = session.createQuery(hql, Producto.class);
        query.setParameter("codigo", codigoBarras);

        // Obtiene el único resultado, "uniqueResult", posible (único porque "codigo_barras" es "UNIQUE").
        Producto producto = query.uniqueResult();

        // Si no hay productos, es decir, si la tabla "productos" está vacía se muestra un mensaje informativo.
        if (producto == null) {
            System.out.println("\n\n\tNo se encontró o no existe ningún productos con el siguiente código de barras (" + codigoBarras + ").");
            return;
        }

        System.out.println("\n\n\tProducto encontrado: " + producto.getNombre() + "\tNúmero de identificación (ID): " + producto.getId() + "\tCódigo de barras: " + producto.get
        );

        // Bloque de código para crear un nuevo lote con fecha de caducidad a 1 año del actual.
        LocalDate fechaCaducidadActual = LocalDate.now().plusYears(yearsToAdd: 1);
        Lote nuevoLote = new Lote(codigo: codigoLote, cantidad: 25, fechaCaducidad: fechaCaducidadActual, estado: EstadoLote.DISPONIBLE);

        // Bloque de código para añadir el nuevo lote creado al producto. Aquí se emplea el método "Helper" para mantener consistencia bidireccional.
        producto.addLote(lote: nuevoLote);
    }
}
```

Operación ‘crear lote/s’ (Método “agregarLoteAProductoExistente(String codigoBarras, String codigoLote)”) (GestionSupermercado.java).

Alumno: Rodrigo López Pérez

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)

4. Paso

“Operaciones Read”.

```
180 // ===== OPERACIONES READ =====
181 // Crea el método "listarTodosLosProveedores". Su función será ir recuperando, leyendo y mostrando todos los objetos, proveedores, de la tabla "proveedores" de nuestra base de datos.
182 // Busca y hace una lista de todos los proveedores registrados en la tabla "proveedores" de nuestra base de datos.
183
184 public void listarTodosLosProveedores() {
185     // Se abre una sesión de Hibernate para interactuar con la base de datos.
186     session = HibernateUtil.getSessionFactory().openSession();
187     //transaction = null;
188
189     System.out.println(":\n\n\tBuscando y listando a todos los proveedpres registrados en la tabla \"proveedores\" de nuestra DB...");
190
191     try {
192         // Bloque de código para buscar todos los proveedores registrados en la DB mediante consulta HQL (Hibernate Query Language).
193         // Forma 1.
194         // List<Proveedor> proveedores = session.createQuery("FROM Proveedor", Proveedor.class).list();
195         // Forma 2.
196         String hql = "FROM Proveedor";
197         Query<Proveedor> query = session.createQuery(string: hql, type: Proveedor.class);
198         List<Proveedor> proveedores = query.list();
199
200         // Si no hay proveedores registrados se muestra un mensaje informativo.
201         if (proveedores.isEmpty()) {
202             System.out.println(":\n\n\tNo hay o no se ha encontrado ningún proveedor. Por el momento no hay proveedores registrados en la base de datos.");
203             return;
204         }
205
206         int contador = 1;
207
208         System.out.println(":\n\n\t(===== LISTA de PROVEEDORES =====>");
209
210         for (Proveedor proveedor : proveedores) {
211             System.out.println("\n\tNúmero de proveedores: " + contador);
212             //System.out.println(proveedor);
213             System.out.println("\n\tProveedor: " + proveedor.getNombre() + " \tCódigo de identificación fiscal (CIF): " + proveedor.getCif() + " \n\t\tContacto:\n\t\t\tCorreo electrónico: " + proveedor.getEmail());
214             //System.out.println(proveedor);
215             System.out.println(":-".repeat(count: 70));
216
217             contador ++;
218         }
219     }
220 }
```

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

```
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help SupermercadoHibernate - Apache NetBeans IDE 20
GestionSupermercado.java x
Source History
228 // Crea el método "buscarProductoPorId". Su función será ir recuperando leyendo el objeto, producto, correspondiente a su número de identificación (ID) de la tabla "prod
229 // Busca un producto por su número de identificación o ID registrado en la base de datos.
230 public void buscarProductoPorId(Long idProducto){
231 // Se abre una sesión de Hibernate para interactuar con la base de datos.
232 session = HibernateUtil.getSessionFactory().openSession();
233 //transaction = null;
234
235 System.out.println("\n\n\tBuscando un producto por su número de identificació (ID) (" + idProducto + "...");
236
237 try{
238 // Bloque de código para buscar un producto por su número de identificación (ID).
239 Producto producto = session.get((Type)Producto.class, ariml: idProducto);
240
241 // Si no hay productos, es decir, si la tabla "productos" está vacía se muestra un mensaje informativo.
242 if (producto == null){
243 System.out.println("\n\n\tNo se encontró o no existe ningún producto con el número de identificación (ID): (" + idProducto + "...");
244 return;
245 }
246
247 System.out.println(s: "\n\n\t<===== PRODUCTO ENCONTRADO =====>");
248
249 System.out.println(s: "\n\n\tNombre: " + producto.getNombre() + " " + "\tNúmero de identificación (ID): " + idProducto + "\tCódigo de barras: " + producto.getCodigoBarras()
250
251 System.out.println(s: "\n\n\t<===== PROVEEDOR DEL PRODUCTO =====>");
252
253 System.out.println(s: "\n\n\tNombre: " + producto.getProveedor().getNombre() + "\tNúmero de identificación (ID): " + producto.getProveedor().getId() + "\tNúmero de identif
254
255 System.out.println(s: "\n\n\t<===== LOTES DEL PROVEEDOR =====>");
256
257 List<Lote> lotes = producto.getLotes();
258 System.out.println("\n\n\tNúmero de productos provistos: " + lotes.size() + " en total.");
259
260 // Si no hay lotes registrados se muestra un mensaje informativo.
261 if (lotes.isEmpty()){
262 System.out.println(s: "\n\n\tNo hay o no se ha encontrado ningún lote. Por el momento no hay lotes registrados para dicho producto en la base de datos.");
263 return;
264 }
265
266 int contador = 1;
```

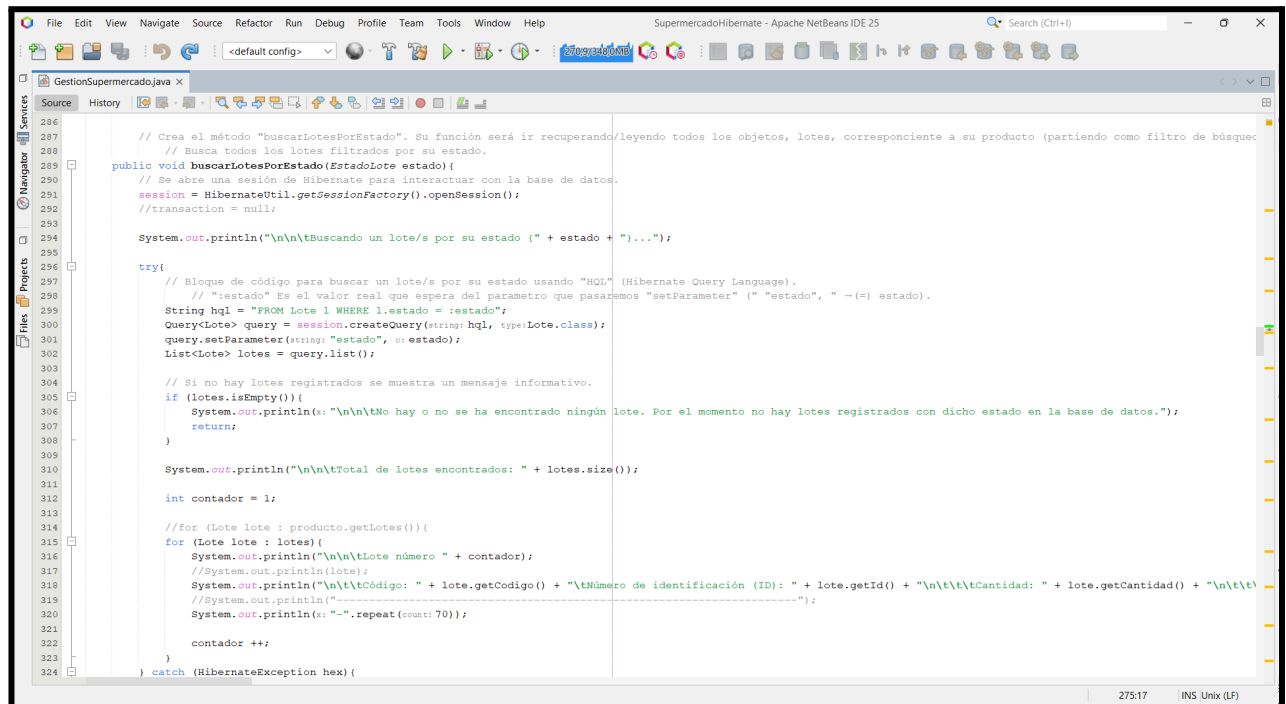
Operación ‘buscar producto’ (Método “buscarProductoPorId(Long idProducto)”) (GestionSupermercado.java).

Alumno: Rodrigo López Pérez

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

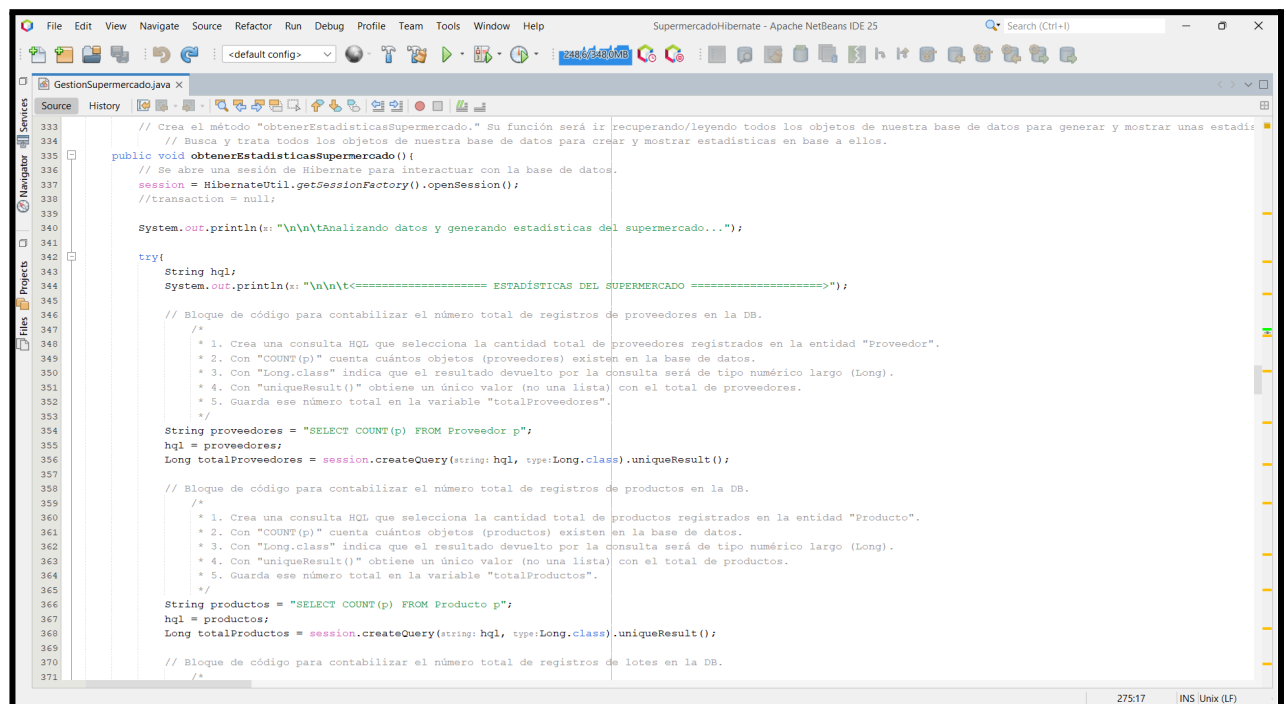
Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)



```
286 // Crea el método "buscarLotesPorEstado". Su función será ir recuperando/leyendo todos los objetos, lotes, correspondiente a su producto (partiendo como filtro de búsqueda)
287 // Busca todos los lotes filtrados por su estado.
288 public void buscarLotesPorEstado(EstadoLote estado) {
289     // Se abre una sesión de Hibernate para interactuar con la base de datos.
290     session = HibernateUtil.getSessionFactory().openSession();
291     //transaction = null;
292
293     System.out.println("\n\n\tBuscando un lote/s por su estado (" + estado + ")...");
294
295     try{
296         // Bloque de código para buscar un lote/s por su estado usando "HQL" (Hibernate Query Language).
297         // "estado" Es el valor real que espera del parametro que pasaremos "setParameter" (" estado", " - (=) estado).
298         String hql = "FROM Lote l WHERE l.estado = :estado";
299         Query<Lote> query = session.createQuery(string: hql, type:Lote.class);
300         query.setParameter(string: "estado", o: estado);
301         List<Lote> lotes = query.list();
302
303         // Si no hay lotes registrados se muestra un mensaje informativo.
304         if (lotes.isEmpty()){
305             System.out.println(s: "\n\n\tNo hay o no se ha encontrado ningún lote. Por el momento no hay lotes registrados con dicho estado en la base de datos.");
306             return;
307         }
308
309         System.out.println("\n\n\tTotal de lotes encontrados: " + lotes.size());
310
311         int contador = 1;
312
313         //for (Lote lote : producto.getLotes()){
314         for (Lote lote : lotes){
315             System.out.println("\n\n\tLote número " + contador);
316             //System.out.println(lote);
317             System.out.println("\n\n\tCódigo: " + lote.getCodigo() + "\n\n\tNúmero de identificación (ID): " + lote.getId() + "\n\n\tCantidad: " + lote.getCantidad() + "\n\n\t");
318             System.out.println("-----");
319             System.out.println(s: "--.repeat(count: 70));
320
321             contador ++;
322         }
323     } catch (HibernateException hex) {
```

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



```
333 // Crea el método "obtenerEstadisticasSupermercado". Su función será ir recuperando/leyendo todos los objetos de nuestra base de datos para generar y mostrar unas estadísticas
334 // Busca y trata todos los objetos de nuestra base de datos para crear y mostrar estadísticas en base a ellos.
335 public void obtenerEstadisticasSupermercado() {
336     // Se abre una sesión de Hibernate para interactuar con la base de datos.
337     session = HibernateUtil.getSessionFactory().openSession();
338     //transaction = null;
339
340     System.out.println(s: "\n\n\tAnalizando datos y generando estadísticas del supermercado...");
341
342     try{
343         String hql;
344         System.out.println(s: "\n\n\t===== ESTADÍSTICAS DEL SUPERMERCADO =====>");
345
346         // Bloque de código para contabilizar el número total de registros de proveedores en la DB.
347         /*
348          * 1. Crea una consulta HQL que selecciona la cantidad total de proveedores registrados en la entidad "Proveedor".
349          * 2. Con "COUNT(p)" cuenta cuántos objetos (proveedores) existen en la base de datos.
350          * 3. Con "Long.class" indica que el resultado devuelto por la consulta será de tipo numérico largo (Long).
351          * 4. Con "uniqueResult()" obtiene un único valor (no una lista) con el total de proveedores.
352          * 5. Guarda ese número total en la variable "totalProveedores".
353          */
354         String proveedores = "SELECT COUNT(p) FROM Proveedor p";
355         hql = proveedores;
356         Long totalProveedores = session.createQuery(string: hql, type:Long.class).uniqueResult();
357
358         // Bloque de código para contabilizar el número total de registros de productos en la DB.
359         /*
360          * 1. Crea una consulta HQL que selecciona la cantidad total de productos registrados en la entidad "Producto".
361          * 2. Con "COUNT(p)" cuenta cuántos objetos (productos) existen en la base de datos.
362          * 3. Con "Long.class" indica que el resultado devuelto por la consulta será de tipo numérico largo (Long).
363          * 4. Con "uniqueResult()" obtiene un único valor (no una lista) con el total de productos.
364          * 5. Guarda ese número total en la variable "totalProductos".
365          */
366         String productos = "SELECT COUNT(p) FROM Producto p";
367         hql = productos;
368         Long totalProductos = session.createQuery(string: hql, type:Long.class).uniqueResult();
369
370         // Bloque de código para contabilizar el número total de registros de lotes en la DB.
371         /*
```

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

Alumno: Rodrigo López Pérez

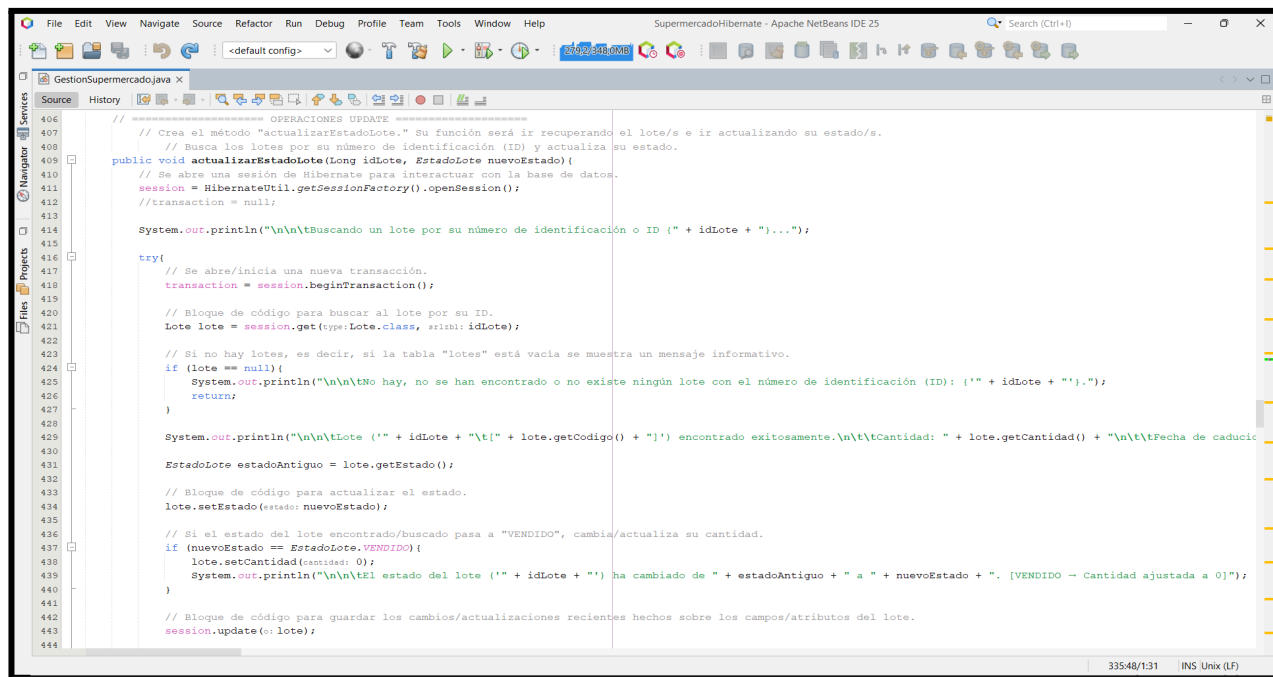
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)

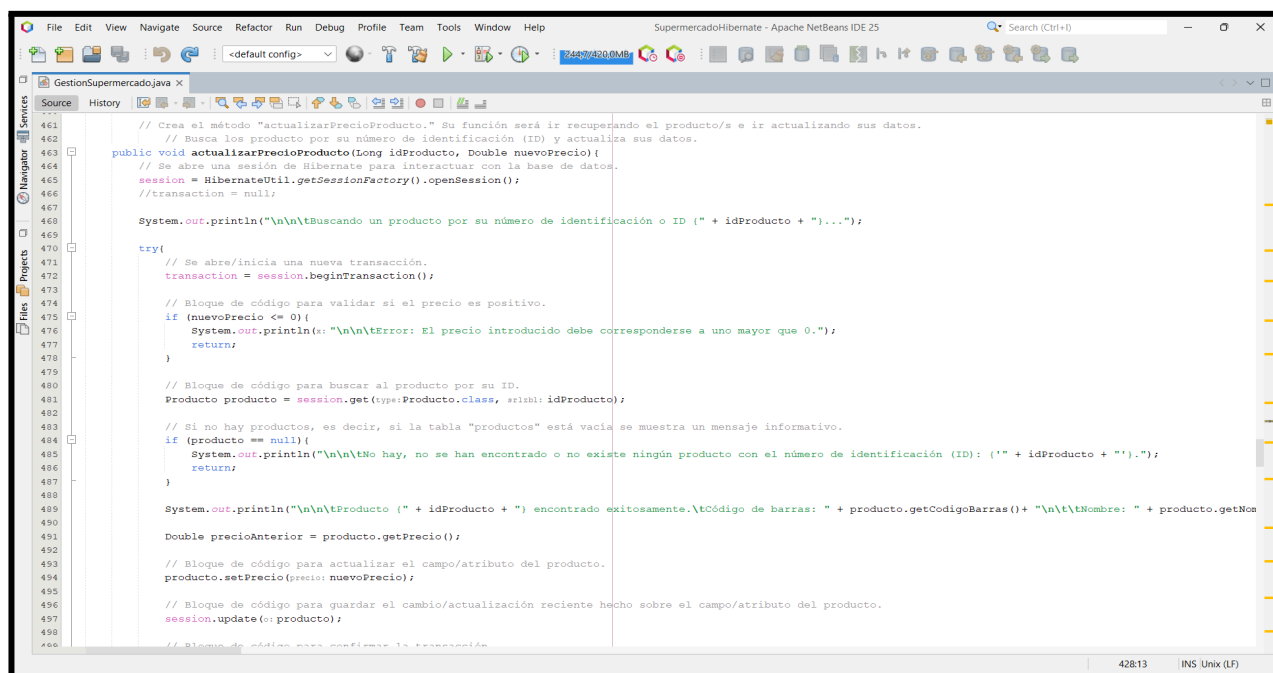
5. Paso

“Operaciones Update”.



```
406 // OPERACIONES UPDATE
407 // Crea el método "actualizarEstadoLote." Su función será ir recuperando el lote/s e ir actualizando su estado/s.
408 // Busca los lotes por su número de identificación (ID) y actualiza su estado.
409 public void actualizarEstadoLote(Long idLote, EstadoLote nuevoEstado){
410     // Se abre una sesión de Hibernate para interactuar con la base de datos.
411     session = HibernateUtil.getSessionFactory().openSession();
412     //transaction = null;
413
414     System.out.println("\n\n\tBuscando un lote por su número de identificación o ID (" + idLote + "...");
415
416     try{
417         // Se abre/inicia una nueva transacción.
418         transaction = session.beginTransaction();
419
420         // Bloque de código para buscar al lote por su ID.
421         Lote lote = session.get(type:Lote.class, idLote);
422
423         // Si no hay lotes, es decir, si la tabla "lotes" está vacía se muestra un mensaje informativo.
424         if (lote == null){
425             System.out.println("\n\n\tNo hay, no se han encontrado o no existe ningún lote con el número de identificación (ID): (" + idLote + "...");
426             return;
427         }
428
429         System.out.println("\n\n\tLote (" + idLote + "\t" + lote.getCodigo() + ") encontrado exitosamente.\n\t\tCantidad: " + lote.getCantidad() + "\n\t\tFecha de caducidad: " + lote.getFechaCaducidad());
430
431         EstadoLote estadoAntiguo = lote.getEstado();
432
433         // Bloque de código para actualizar el estado.
434         lote.setEstado(estado: nuevoEstado);
435
436         // Si el estado del lote encontrado/buscado pasa a "VENDIDO", cambia/actualiza su cantidad.
437         if (nuevoEstado == EstadoLote.VENDIDO){
438             lote.setCantidad(cantidad: 0);
439             System.out.println("\n\n\tEl estado del lote (" + idLote + ") ha cambiado de " + estadoAntiguo + " a " + nuevoEstado + ". [VENDIDO -> Cantidad ajustada a 0]");
440         }
441
442         // Bloque de código para guardar los cambios/actualizaciones recientes hechos sobre los campos/atributos del lote.
443         session.update(lote);
444     }
```

Operación ‘actualizar estado lote/s’ (Método “actualizarEstadoLote(Long idLote, EstadoLote nuevoEstado)”) (GestionSupermercado.java).



```
441 // Crea el método "actualizarPrecioProducto." Su función será ir recuperando el producto/s e ir actualizando sus datos.
442 // Busca los productos por su número de identificación (ID) y actualiza sus datos.
443 public void actualizarPrecioProducto(Long idProducto, Double nuevoPrecio){
444     // Se abre una sesión de Hibernate para interactuar con la base de datos.
445     session = HibernateUtil.getSessionFactory().openSession();
446     //transaction = null;
447
448     System.out.println("\n\n\tBuscando un producto por su número de identificación o ID (" + idProducto + "...");
449
450     try{
451         // Se abre/inicia una nueva transacción.
452         transaction = session.beginTransaction();
453
454         // Bloque de código para validar si el precio es positivo.
455         if (nuevoPrecio <= 0){
456             System.out.println("\n\n\tError: El precio introducido debe corresponderse a uno mayor que 0.");
457             return;
458         }
459
460         // Bloque de código para buscar al producto por su ID.
461         Producto producto = session.get(type:Producto.class, idProducto);
462
463         // Si no hay productos, es decir, si la tabla "productos" está vacía se muestra un mensaje informativo.
464         if (producto == null){
465             System.out.println("\n\n\tNo hay, no se han encontrado o no existe ningún producto con el número de identificación (ID): (" + idProducto + "...");
466             return;
467         }
468
469         System.out.println("\n\n\tProducto (" + idProducto + ") encontrado exitosamente.\n\t\tCódigo de barras: " + producto.getCodigoBarras() + "\n\t\tNombre: " + producto.getNombre());
470
471         Double precioAnterior = producto.getPrecio();
472
473         // Bloque de código para actualizar el campo/atributo del producto.
474         producto.setPrecio(precio: nuevoPrecio);
475
476         // Bloque de código para guardar el cambio/actualización reciente hecho sobre el campo/atributo del producto.
477         session.update(producto);
478     }
```

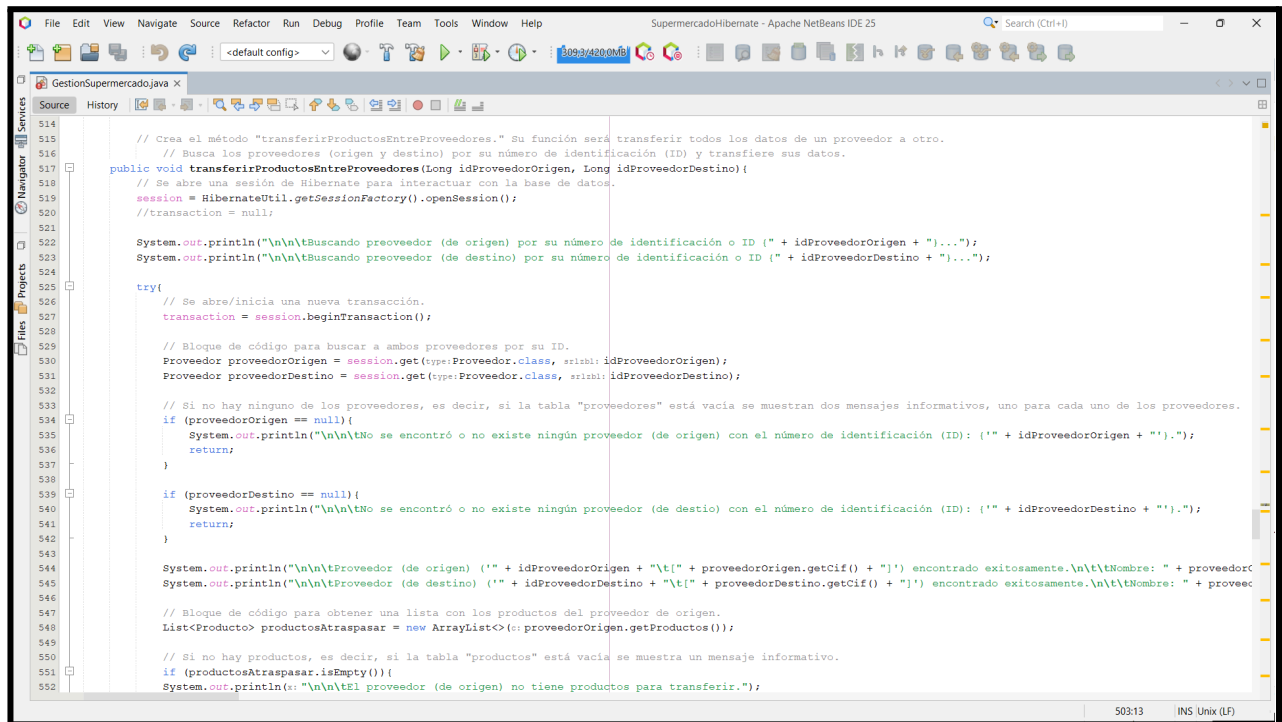
Operación ‘actualizar precio producto/s’ (Método “actualizarPrecioProducto(Long idProducto, Double nuevoPrecio)”) (GestionSupermercado.java).

Alumno: Rodrigo López Pérez

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)



```
514 // Crea el método "transferirProductosEntreProveedores." Su función será transferir todos los datos de un proveedor a otro.
515 // Busca los proveedores (origen y destino) por su número de identificación (ID) y transfiere sus datos.
516 public void transferirProductosEntreProveedores(Long idProveedorOrigen, Long idProveedorDestino){
517 // Se abre una sesión de Hibernate para interactuar con la base de datos.
518 session = HibernateUtil.getSessionFactory().openSession();
519 //transaction = null;
520
521 System.out.println("\n\n\tBuscando proveedor (de origen) por su número de identificación o ID (" + idProveedorOrigen + "...");
522 System.out.println("\n\n\tBuscando proveedor (de destino) por su número de identificación o ID (" + idProveedorDestino + "...");
523
524 try{
525 // Se abre/inicia una nueva transacción.
526 transaction = session.beginTransaction();
527
528 // Bloque de código para buscar a ambos proveedores por su ID.
529 Proveedor proveedorOrigen = session.get(type:Proveedor.class, serial: idProveedorOrigen);
530 Proveedor proveedorDestino = session.get(type:Proveedor.class, serial: idProveedorDestino);
531
532 // Si no hay ninguno de los proveedores, es decir, si la tabla "proveedores" está vacía se muestran dos mensajes informativos, uno para cada uno de los proveedores.
533 if (proveedorOrigen == null){
534 System.out.println("\n\n\tNo se encontró o no existe ningún proveedor (de origen) con el número de identificación (ID): (" + idProveedorOrigen + "...");
535 return;
536 }
537
538 if (proveedorDestino == null){
539 System.out.println("\n\n\tNo se encontró o no existe ningún proveedor (de destino) con el número de identificación (ID): (" + idProveedorDestino + "...");
540 return;
541 }
542
543 System.out.println("\n\n\tProveedor (de origen) (" + idProveedorOrigen + "\t[" + proveedorOrigen.getCif() + "]) encontrado exitosamente.\n\n\tNombre: " + proveedorOrigen.getNombre();
544 System.out.println("\n\n\tProveedor (de destino) (" + idProveedorDestino + "\t[" + proveedorDestino.getCif() + "]) encontrado exitosamente.\n\n\tNombre: " + proveedorDestino.getNombre();
545
546 // Bloque de código para obtener una lista con los productos del proveedor de origen.
547 List<Producto> productosAtraspasar = new ArrayList<>(: proveedorOrigen.getProductos());
548
549 // Si no hay productos, es decir, si la tabla "productos" está vacía se muestra un mensaje informativo.
550 if (productosAtraspasar.isEmpty()){
551 System.out.println("\n\n\tEl proveedor (de origen) no tiene productos para transferir.");
552 }
```

Operación ‘transferir productos entre proveedores’ (Método “transferirProductosEntreProveedores(Long idProveedorOrigen, Long idProveedorDestino)”) (GestionSupermercado.java).

Alumno: Rodrigo López Pérez

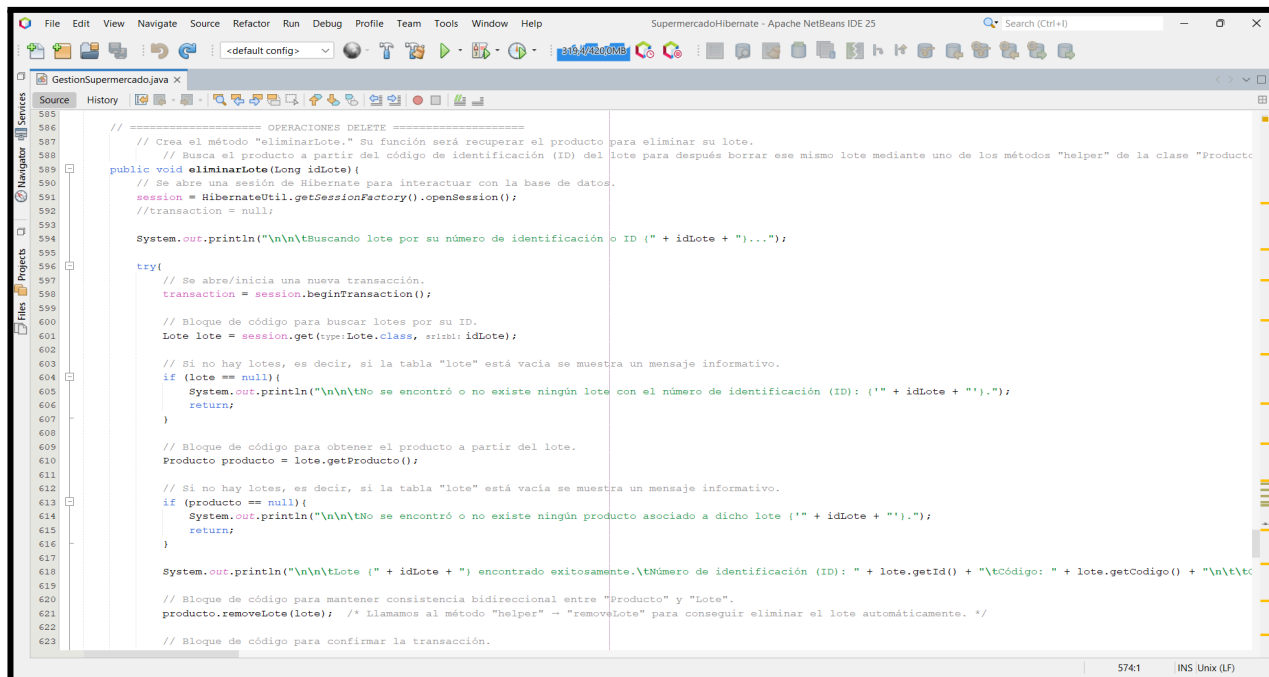
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)

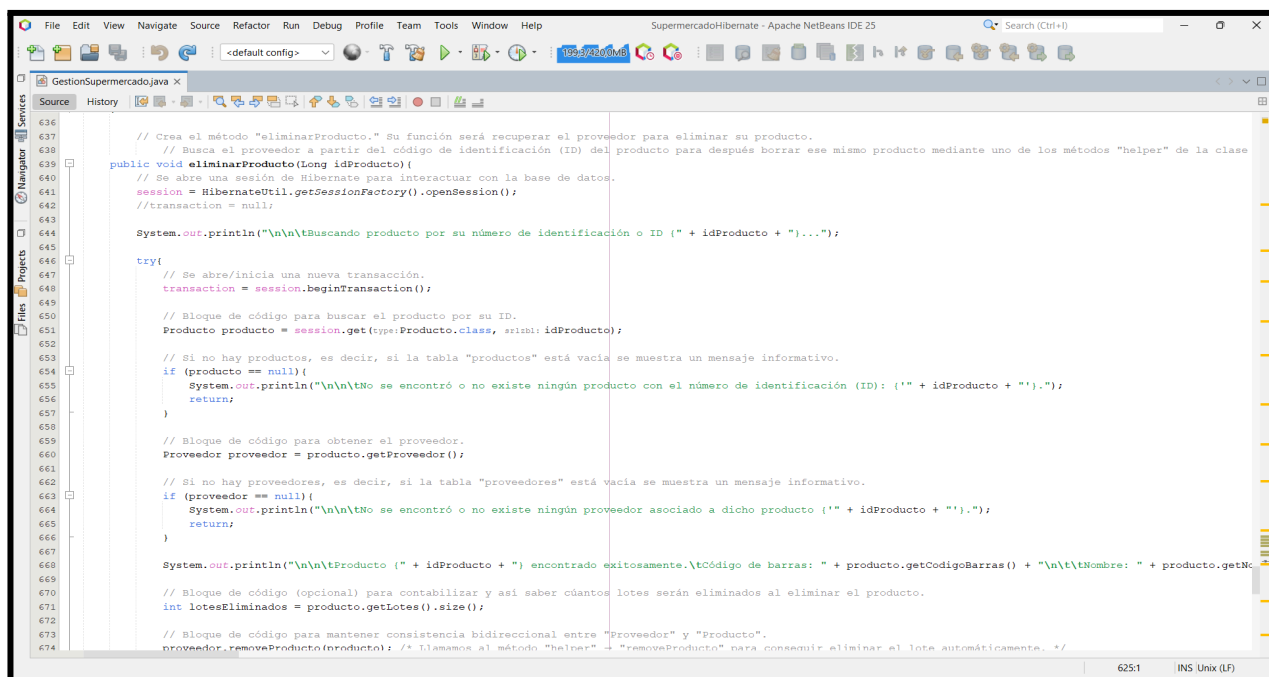
6. Paso

“Operaciones Delete”.



```
585 // ===== OPERACIONES DELETE =====
586 // Crea el método "eliminarLote." Su función será recuperar el producto para eliminar su lote.
587 // Busca el producto a partir del código de identificación (ID) del lote para después borrar ese mismo lote mediante uno de los métodos "helper" de la clase "Producto"
588 public void eliminarLote(Long idLote) {
589     // Se abre una sesión de Hibernate para interactuar con la base de datos.
590     session = HibernateUtil.getSessionFactory().openSession();
591     //transaction = null;
592
593     System.out.println("\n\n\tBuscando lote por su número de identificación o ID (" + idLote + "...");
594
595     try {
596         // Se abre/inicia una nueva transacción.
597         transaction = session.beginTransaction();
598
599         // Bloque de código para buscar lotes por su ID.
600         Lote lote = session.get(type:Lote.class, idLote);
601
602         // Si no hay lotes, es decir, si la tabla "lote" está vacía se muestra un mensaje informativo.
603         if (lote == null) {
604             System.out.println("\n\n\tNo se encontró o no existe ningún lote con el número de identificación (ID): (" + idLote + "...");
605             return;
606         }
607
608         // Bloque de código para obtener el producto a partir del lote.
609         Producto producto = lote.getProducto();
610
611         // Si no hay lotes, es decir, si la tabla "lote" está vacía se muestra un mensaje informativo.
612         if (producto == null) {
613             System.out.println("\n\n\tNo se encontró o no existe ningún producto asociado a dicho lote (" + idLote + "...");
614             return;
615         }
616
617         System.out.println("\n\n\tLote (" + idLote + ") encontrado exitosamente.\tNúmero de identificación (ID): " + lote.getId() + "\tCódigo: " + lote.getCodigo() + "\n\n\t");
618
619         // Bloque de código para mantener consistencia bidireccional entre "Producto" y "Lote".
620         producto.removeLote(lote); /* Llamamos al método "helper" -> "removeLote" para conseguir eliminar el lote automáticamente. */
621
622         // Bloque de código para confirmar la transacción.
623         transaction.commit();
624     } catch (Exception e) {
625         System.out.println("\n\n\tError al eliminar el lote: " + e.getMessage());
626     }
627 }
```

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



```
636 // Crea el método "eliminarProducto." Su función será recuperar el proveedor para eliminar su producto.
637 // Busca el proveedor a partir del código de identificación (ID) del producto para después borrar ese mismo producto mediante uno de los métodos "helper" de la clase "Producto"
638 public void eliminarProducto(Long idProducto) {
639     // Se abre una sesión de Hibernate para interactuar con la base de datos.
640     session = HibernateUtil.getSessionFactory().openSession();
641     //transaction = null;
642
643     System.out.println("\n\n\tBuscando producto por su número de identificación o ID (" + idProducto + "...");
644
645     try {
646         // Se abre/inicia una nueva transacción.
647         transaction = session.beginTransaction();
648
649         // Bloque de código para buscar el producto por su ID.
650         Producto producto = session.get(type:Producto.class, idProducto);
651
652         // Si no hay productos, es decir, si la tabla "productos" está vacía se muestra un mensaje informativo.
653         if (producto == null) {
654             System.out.println("\n\n\tNo se encontró o no existe ningún producto con el número de identificación (ID): (" + idProducto + "...");
655             return;
656         }
657
658         // Bloque de código para obtener el proveedor.
659         Proveedor proveedor = producto.getProveedor();
660
661         // Si no hay proveedores, es decir, si la tabla "proveedores" está vacía se muestra un mensaje informativo.
662         if (proveedor == null) {
663             System.out.println("\n\n\tNo se encontró o no existe ningún proveedor asociado a dicho producto (" + idProducto + "...");
664             return;
665         }
666
667         System.out.println("\n\n\tProducto (" + idProducto + ") encontrado exitosamente.\tCódigo de barras: " + producto.getCodigoBarras() + "\n\n\tNombre: " + producto.getNombre() + "\n\n\t");
668
669         // Bloque de código (opcional) para contabilizar y así saber cuántos lotes serán eliminados al eliminar el producto.
670         int lotesEliminados = producto.getLotes().size();
671
672         // Bloque de código para mantener consistencia bidireccional entre "Proveedor" y "Producto".
673         proveedor.removeProducto(producto); /* Llamamos al método "helper" -> "removeProducto" para conseguir eliminar el lote automáticamente. */
674
675         // Bloque de código para confirmar la transacción.
676         transaction.commit();
677     } catch (Exception e) {
678         System.out.println("\n\n\tError al eliminar el producto: " + e.getMessage());
679     }
680 }
```

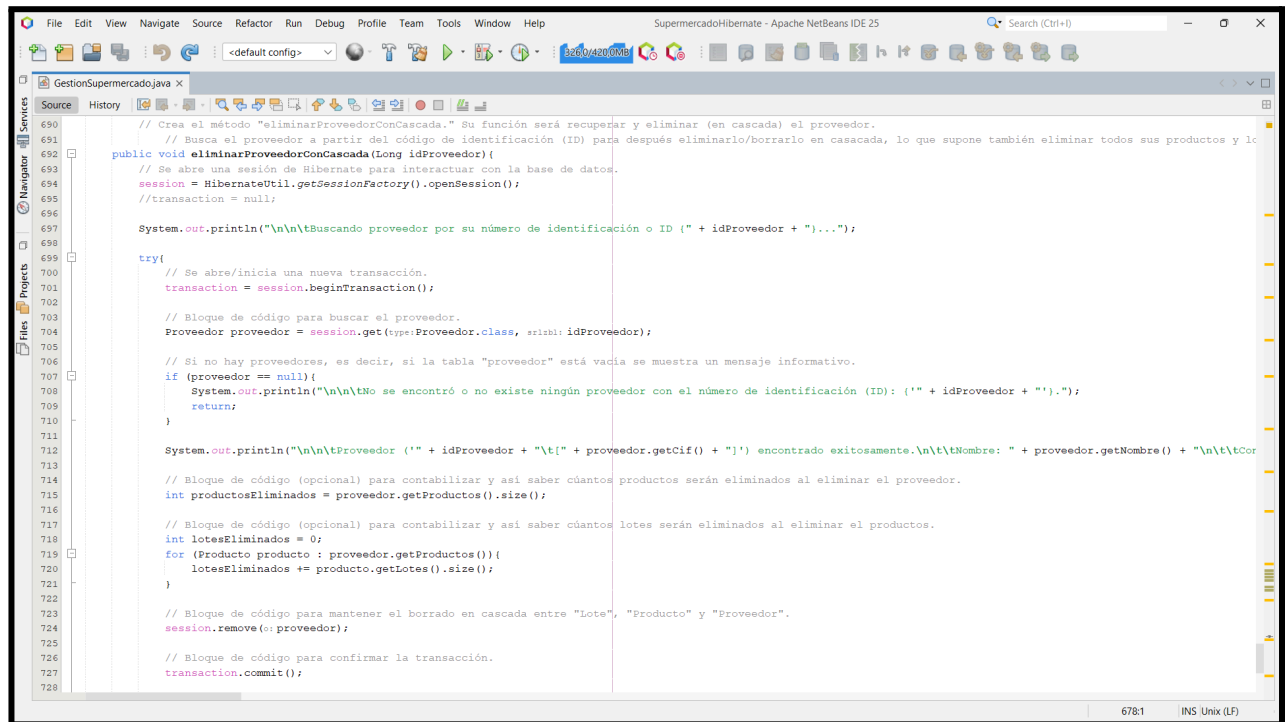
Operación 'eliminar producto/s' (Método "eliminarProducto(Long idProducto)") (GestionSupermercado.java).

Alumno: Rodrigo López Pérez

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)



```
690 // Crea el método "eliminarProveedorConCascada." Su función será recuperar y eliminar (en cascada) el proveedor.
691 // Busca el proveedor a partir del código de identificación (ID) para después eliminarlo/borrarlo en cascada, lo que supone también eliminar todos sus productos y l
692 public void eliminarProveedorConCascada(Long idProveedor) {
693     // Se abre una sesión de Hibernate para interactuar con la base de datos.
694     session = HibernateUtil.getSessionFactory().openSession();
695     //transaction = null;
696
697     System.out.println("\n\nBuscando proveedor por su número de identificación o ID (" + idProveedor + "...");
698
699     try {
700         // Se abre/inicia una nueva transacción.
701         transaction = session.beginTransaction();
702
703         // Bloque de código para buscar el proveedor.
704         Proveedor proveedor = session.get(type:Proveedor.class, id=idProveedor);
705
706         // Si no hay proveedores, es decir, si la tabla "proveedor" está vacía se muestra un mensaje informativo.
707         if (proveedor == null) {
708             System.out.println("\n\nNo se encontró o no existe ningún proveedor con el número de identificación (ID): (" + idProveedor + "...");
709             return;
710         }
711
712         System.out.println("\n\nProveedor (" + idProveedor + " [" + proveedor.getCif() + "]) encontrado exitosamente.\n\n\tNombre: " + proveedor.getNombre() + "\n\n\tCor
713
714         // Bloque de código (opcional) para contabilizar y así saber cuántos productos serán eliminados al eliminar el proveedor.
715         int productosEliminados = proveedor.getProductos().size();
716
717         // Bloque de código (opcional) para contabilizar y así saber cuántos lotes serán eliminados al eliminar el productos.
718         int lotesEliminados = 0;
719         for (Producto producto : proveedor.getProductos()) {
720             lotesEliminados += producto.getLotes().size();
721         }
722
723         // Bloque de código para mantener el borrado en cascada entre "Lote", "Producto" y "Proveedor".
724         session.remove(proveedor);
725
726         // Bloque de código para confirmar la transacción.
727         transaction.commit();
728     }
```

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

Alumno: Rodrigo López Pérez

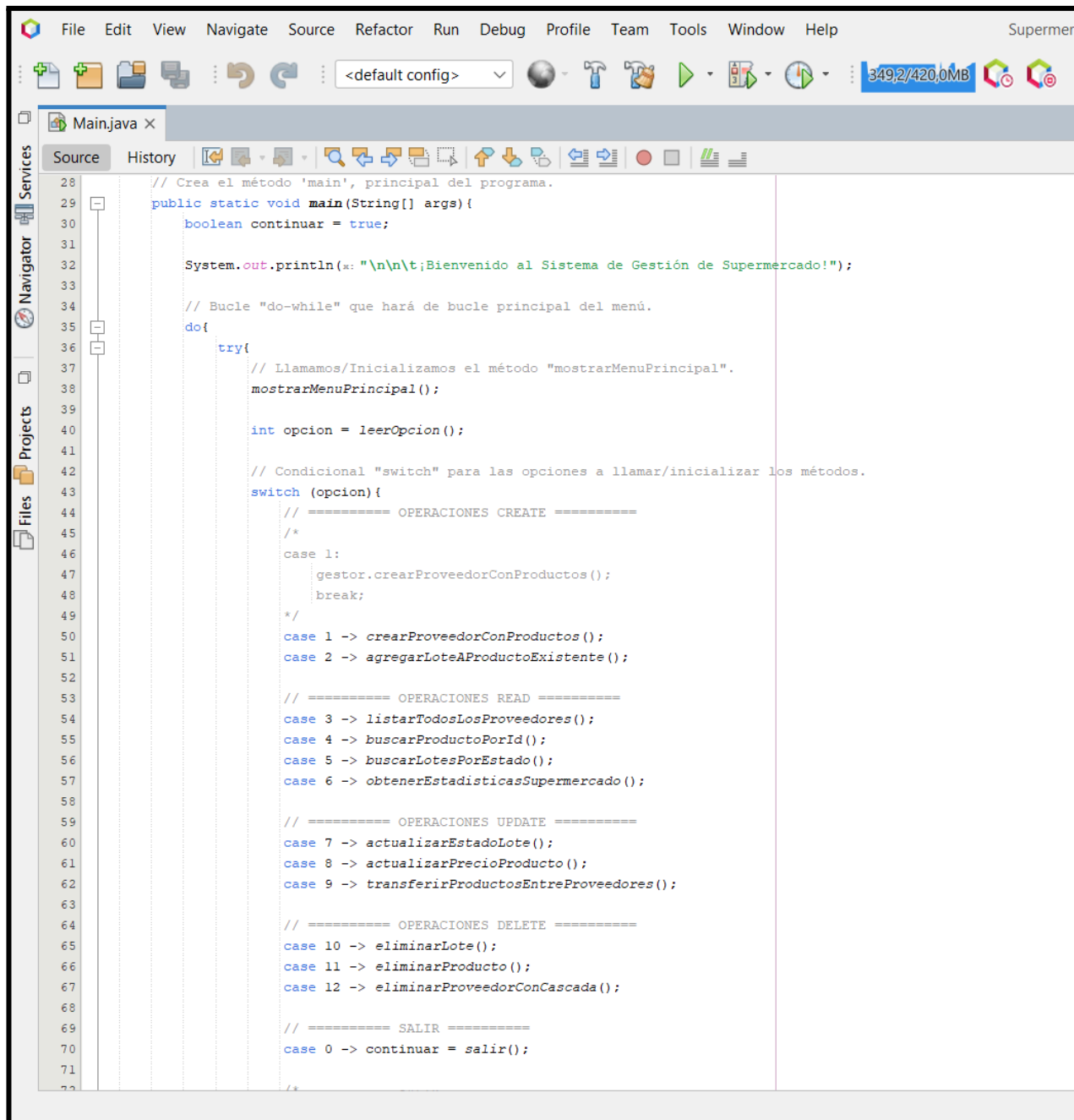
Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)

7. Paso

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



```
28 // Crea el método 'main', principal del programa.
29 public static void main(String[] args){
30     boolean continuar = true;
31
32     System.out.println("\n\n\tBienvenido al Sistema de Gestión de Supermercado!");
33
34     // Bucle "do-while" que hará de bucle principal del menú.
35     do{
36         try{
37             // Llamamos/Inicializamos el método "mostrarMenuPrincipal".
38             mostrarMenuPrincipal();
39
40             int opcion = leerOpcion();
41
42             // Condicional "switch" para las opciones a llamar/inicializar los métodos.
43             switch (opcion){
44                 // ===== OPERACIONES CREATE =====
45                 /*
46                 case 1:
47                     gestor.crearProveedorConProductos();
48                     break;
49                 */
50                 case 1 -> crearProveedorConProductos();
51                 case 2 -> agregarLoteAProductoExistente();
52
53                 // ===== OPERACIONES READ =====
54                 case 3 -> listarTodosLosProveedores();
55                 case 4 -> buscarProductoPorId();
56                 case 5 -> buscarLotesPorEstado();
57                 case 6 -> obtenerEstadisticasSupermercado();
58
59                 // ===== OPERACIONES UPDATE =====
60                 case 7 -> actualizarEstadoLote();
61                 case 8 -> actualizarPrecioProducto();
62                 case 9 -> transferirProductosEntreProveedores();
63
64                 // ===== OPERACIONES DELETE =====
65                 case 10 -> eliminarLote();
66                 case 11 -> eliminarProducto();
67                 case 12 -> eliminarProveedorConCascada();
68
69                 // ===== SALIR =====
70                 case 0 -> continuar = salir();
71
72             }
73         }
```

Alumno: Rodrigo López Pérez

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

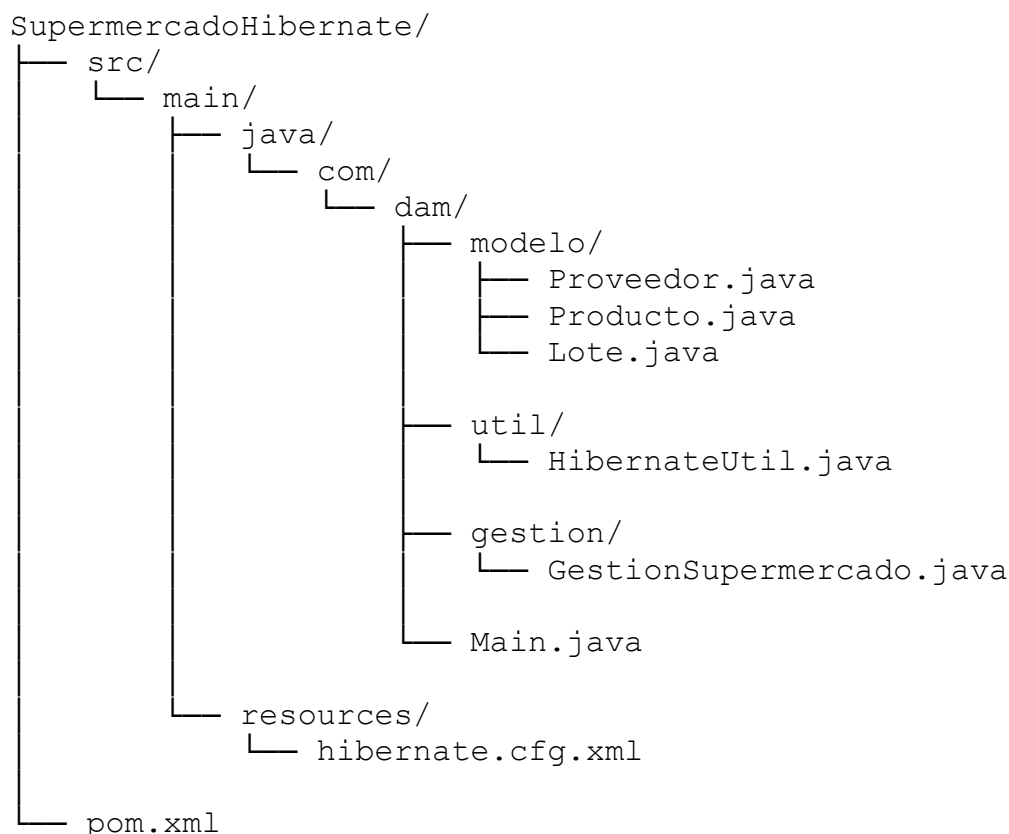
Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)

8. Paso

“Explicación del programa”.

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



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.

Alumno: Rodrigo López Pérez

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)

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.

Alumno: Rodrigo López Pérez

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)

- “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()”.

Alumno: Rodrigo López Pérez

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.

Alumno: Rodrigo López Pérez

Actividad recuperación 3.10 (AED) - Actividad: Supermercado

Profesor: Javier Sebastián Herrero

Módulo: Acceso a datos (AED)

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”.

Alumno: Rodrigo López Pérez