Resolución Trabajo Práctico 10: Acceso a Base de Datos con JDBC

En este trabajo desarrollamos un sistema básico de gestión de categorías, productos y pedidos utilizando Java y bases de datos relacionales. Implementamos un CRUD completo para las entidades principales: categorías, productos y pedidos, con sus respectivos detalles. Para cada entidad se crearon modelos, DAOs y servicios que manejan la lógica de negocio y validaciones.

Se utilizaron transacciones para asegurar la consistencia de los datos, especialmente al momento de crear pedidos con múltiples productos, validando el stock disponible y aplicando rollback en caso de errores. Además, se implementó la visualización detallada de los pedidos, mostrando productos, categorías, cantidades y subtotales.

Codigo:
Modelos:
Categoria:

```java
public class Categoria {
    // Declaramos atributos
    private int id;
    private String nombre;
    private String descripcion;

    // Creamos el constructor
    public Categoria(int id, String nombre, String descripcion) {
        this.id = id;
        this.nombre = nombre;
        this.descripcion = descripcion;
    }

    // Creamos metodos getter y setter
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

}
```

Producto:

```java
public class Producto {
    // Declaramos atributos
    private int id;
    private String nombre;
    private String descripcion;
    private double precio;
    private int cantidad;
    private int idCategoria;

    // Creamos los constructores
    public Producto(String nombre, String descripcion, double precio, int cantidad, int idCategoria) {
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.precio = precio;
        this.cantidad = cantidad;
        this.idCategoria = idCategoria;
    }

    public Producto(int id, String nombre, String descripcion, double precio, int cantidad, int idCategoria) {
        this.id = id;
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.precio = precio;
        this.cantidad = cantidad;
        this.idCategoria = idCategoria;
    }


    // Creamos los getter y setter
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getDescripcion() {
        return descripcion;
    }

    public void setDescripcion(String descripcion) {
        this.descripcion = descripcion;
    }

    public double getPrecio() {
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }

    public int getCantidad() {
        return cantidad;
    }

    public void setCantidad(int cantidad) {
        this.cantidad = cantidad;
    }

    public int getIdCategoria() {
        return idCategoria;
    }

    public void setIdCategoria(int idCategoria) {
        this.idCategoria = idCategoria;
    }
```

Item pedido:

```java
public class ItemPedido {
    // Declaramos los atributos
    private int id;
    private int idPedido;
    private int idProducto;
    private int cantidad;
    private double subtotal;

    // Declaramos el constructor
    public ItemPedido(int id, int idPedido, int idProducto, int cantidad, double subtotal) {
        this.id = id;
        this.idPedido = idPedido;
        this.idProducto = idProducto;
        this.cantidad = cantidad;
        this.subtotal = subtotal;
    }

    // Declaramos los metodos getter y setter
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getIdPedido() {
        return idPedido;
    }

    public void setIdPedido(int idPedido) {
        this.idPedido = idPedido;
    }

    public int getIdProducto() {
        return idProducto;
    }

    public void setIdProducto(int idProducto) {
        this.idProducto = idProducto;
    }

    public int getCantidad() {
        return cantidad;
    }

    public void setCantidad(int cantidad) {
        this.cantidad = cantidad;
    }

    public double getSubtotal() {
        return subtotal;
    }

    public void setSubtotal(double subtotal) {
        this.subtotal = subtotal;
    }


}
```

Pedido:

```java
public class Pedido {
    //Declaramos los atributos
    private int id;
    private Date fecha;
    private double total;

    // Creamos el constrcutor
    public Pedido(int id, Date fecha, double total) {
        this.id = id;
        this.fecha = fecha;
        this.total = total;
    }

    public Pedido() {

    }

    // Creamos los getter y setter
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public Date getFecha() {
        return fecha;
    }

    public void setFecha(Date fecha) {
        this.fecha = fecha;
    }

    public double getTotal() {
        return total;
    }

    public void setTotal(double total) {
        this.total = total;
    }

}
```

DAOs:

Generic DAO: cada DAO va a implementar esta interface.

```java
public interface GenericDAO<T> { //Lo declaramos como interface

    // Declaramos los metodos abstractos
    void crear(T entity, Connection conn) throws Exception;

    T leer(int id, Connection conn) throws Exception;

    List<T> listar(Connection conn) throws Exception;

    void actualizar(T entity, Connection conn) throws Exception;

    void eliminar(int id, Connection conn) throws Exception;
}
```

CategoriaDAOImpl:

```java
public class CategoriaDAOImpl implements GenericDAO<Categoria> {

    /**
     * Inserta una nueva categoria en la base de datos.
     *
     * @param categoria Objeto Categoria que contiene los datos a insertar.
     * @param conn Conexión activa a la base de datos.
     * @throws Exception Si ocurre algún error durante la ejecución SQL.
     */
    @Override
    public void crear(Categoria categoria, Connection conn) throws Exception {
        String sql = "INSERT INTO categorias (nombre, descripcion) VALUES (?, ?)";

        try (PreparedStatement stmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
            stmt.setString(1, categoria.getNombre());
            stmt.setString(2, categoria.getDescripcion());
            stmt.executeUpdate();

            try (ResultSet rs = stmt.getGeneratedKeys()) {
                if (rs.next()) {
                    categoria.setId(rs.getInt(1));
                }
            }
        }
    }

    /**
     * Lee y devuelve una categoria a partir de su ID.
     *
     * @param id Identificador único de la categoria.
     * @param conn Conexión activa a la base de datos.
     * @return La categoria encontrada o null si no existe.
     * @throws Exception Si ocurre algún error durante la ejecución SQL.
     */
    @Override
    public Categoria leer(int id, Connection conn) throws Exception {
        String sql = "SELECT * FROM categorias WHERE id = ?";

        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setInt(1, id);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return new Categoria(rs.getInt("id"), rs.getString("nombre"), rs.getString("descripcion"));
                }
            }
        }
        return null;
    }
```

```java
/**
 * Obtiene una lista con todas las categorías almacenadas en la base de
 * datos.
 *
 * @param conn Conexión activa a la base de datos.
 * @return Lista de categorías.
 * @throws Exception Si ocurre algún error durante la ejecución SQL.
 */
@Override
public List<Categoria> listar(Connection conn) throws Exception {
    List<Categoria> lista = new ArrayList<>();
    String sql = "SELECT * FROM categorias";

    try (PreparedStatement stmt = conn.prepareStatement(sql); ResultSet rs = stmt.executeQuery()) {

        while (rs.next()) {
            Categoria c = new Categoria(rs.getInt("id"), rs.getString("nombre"), rs.getString("descripcion'
            lista.add(c);
        }
    }

    return lista;
}

/**
 * Actualiza los datos de una categoría existente en la base de datos.
 *
 * @param categoria Objeto Categoria con los datos actualizados (debe
 * incluir el id).
 * @param conn Conexión activa a la base de datos.
 * @throws Exception Si ocurre algún error durante la ejecución SQL.
 */
@Override
public void actualizar(Categoria categoria, Connection conn) throws Exception {
    String sql = "UPDATE categorias SET nombre = ?, descripcion = ? WHERE id = ?";

    try (PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setString(1, categoria.getNombre());
        stmt.setString(2, categoria.getDescripcion());
        stmt.setInt(3, categoria.getId());
        stmt.executeUpdate();
    }
}
```

```java
    /**
     * Elimina una categoría de la base de datos por su ID.
     *
     * @param id Identificador único de la categoría a eliminar.
     * @param conn Conexión activa a la base de datos.
     * @throws Exception Si ocurre algún error durante la ejecución SQL.
     */
    @Override
    public void eliminar(int id, Connection conn) throws Exception {
        String sql = "DELETE FROM categorias WHERE id = ?";

        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setInt(1, id);
            stmt.executeUpdate();
        }
    }

    /**
     * Verifica si ya existe una categoría con el nombre especificado.
     *
     * @param nombre Nombre de la categoría a verificar.
     * @param conn Conexión activa a la base de datos.
     * @return true si existe una categoría con ese nombre, false en caso
     * contrario.
     * @throws Exception Si ocurre algún error durante la ejecución SQL.
     */
    public boolean existeNombre(String nombre, Connection conn) throws Exception {
        String sql = "SELECT COUNT(*) FROM categorias WHERE nombre = ?";
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setString(1, nombre);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return rs.getInt(1) > 0;
                }
            }
        }
        return false;
    }

}
```

ProductoDAOImpl:

```java
public class ProductoDAOImpl implements GenericDAO<Producto> {

    /**
     * Inserta un nuevo producto en la base de datos.
     *
     * @param producto Objeto Producto con los datos a insertar.
     * @param conn Conexión activa a la base de datos.
     * @throws Exception Si ocurre un error en la ejecución SQL.
     */
    @Override
    public void crear(Producto producto, Connection conn) throws Exception {
        String sql = "INSERT INTO productos (nombre, descripcion, precio, cantidad, id_categoria) values(?, ?, ?, ?, ?)";

        try (PreparedStatement stmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
            stmt.setString(1, producto.getNombre());
            stmt.setString(2, producto.getDescripcion());
            stmt.setDouble(3, producto.getPrecio());
            stmt.setInt(4, producto.getCantidad());
            stmt.setInt(5, producto.getIdCategoria());
            stmt.executeUpdate();

            try (ResultSet rs = stmt.getGeneratedKeys()) {
                if (rs.next()) {
                    producto.setId(rs.getInt(1));
                }
            }
        }
    }
```

```java
/**
 * Obtiene un producto de la base de datos por su ID.
 *
 * @param id Identificador único del producto.
 * @param conn Conexión activa a la base de datos.
 * @return Producto encontrado o null si no existe.
 * @throws Exception Si ocurre un error en la ejecución SQL.
 */
@Override
public Producto leer(int id, Connection conn) throws Exception {
    String sql = "SELECT * FROM productos WHERE id = ?";

    try (PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt(1, id);
        ResultSet rs = stmt.executeQuery();
        if (rs.next()) {
            Integer idCat = rs.getObject("id_categoria") != null ? rs.getInt("id_categoria") : null;
            return new Producto(
                    rs.getInt("id"),
                    rs.getString("nombre"),
                    rs.getString("descripcion"),
                    rs.getDouble("precio"),
                    rs.getInt("cantidad"),
                    idCat
            );
        }
    }
    return null;
}
```

```java
/**
 * Lista todos los productos existentes en la base de datos.
 *
 * @param conn Conexión activa a la base de datos.
 * @return Lista de productos.
 * @throws Exception Si ocurre un error en la ejecución SQL.
 */
@Override
public List<Producto> listar(Connection conn) throws Exception {
    String sql = "SELECT * FROM productos";
    List<Producto> productos = new ArrayList<>();

    try (PreparedStatement stmt = conn.prepareStatement(sql); ResultSet rs = stmt.executeQuery()) {

        while (rs.next()) {
            Integer idCat = rs.getObject("id_categoria") != null ? rs.getInt("id_categoria") : null;
            productos.add(new Producto(
                    rs.getInt("id"),
                    rs.getString("nombre"),
                    rs.getString("descripcion"),
                    rs.getDouble("precio"),
                    rs.getInt("cantidad"),
                    idCat
            ));
        }
    }
    return productos;
}
```

```java
/**
 * Actualiza los datos de un producto existente.
 *
 * @param producto Producto con los datos actualizados (debe incluir el id).
 * @param conn Conexión activa a la base de datos.
 * @throws Exception Si ocurre un error en la ejecución SQL.
 */
@Override
public void actualizar(Producto producto, Connection conn) throws Exception {
    String sql = "UPDATE productos SET nombre = ?, descripcion = ?, precio = ?, cantidad = ?, id_categoria = ? WHERE id = ?";

    try (PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setString(1, producto.getNombre());
        stmt.setString(2, producto.getDescripcion());
        stmt.setDouble(3, producto.getPrecio());
        stmt.setInt(4, producto.getCantidad());
        stmt.setInt(5, producto.getIdCategoria());
        stmt.setInt(6, producto.getId());

        stmt.executeUpdate();
    }
}

/**
 * Elimina un producto de la base de datos por su ID.
 *
 * @param id Identificador del producto a eliminar.
 * @param conn Conexión activa a la base de datos.
 * @throws Exception Si ocurre un error en la ejecución SQL.
 */
@Override
public void eliminar(int id, Connection conn) throws Exception {
    String sql = "DELETE FROM productos WHERE id = ?";
    try (PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt(1, id);
        stmt.executeUpdate();
    }
}
```

```java
/**
 * Obtiene una lista de productos filtrados por categoría.
 *
 * @param conn Conexión activa a la base de datos.
 * @param idCategoria ID de la categoría para filtrar productos.
 * @return Lista de productos que pertenecen a la categoría dada.
 * @throws Exception Si ocurre un error en la ejecución SQL.
 */
public List<Producto> listarPorCategoria(Connection conn, int idCategoria) throws Exception {
    String sql = "SELECT * FROM productos WHERE id_categoria = ?";
    List<Producto> productos = new ArrayList<>();

    try (PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt(1, idCategoria);
        try (ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                productos.add(new Producto(
                        rs.getInt("id"),
                        rs.getString("nombre"),
                        rs.getString("descripcion"),
                        rs.getDouble("precio"),
                        rs.getInt("cantidad"),
                        rs.getInt("id_categoria")
                ));
            }
        }
    }
    return productos;
}
```

```java
/**
 * Verifica si existe una categoría con el ID especificado.
 *
 * @param conn Conexión activa a la base de datos.
 * @param idCategoria ID de la categoría a verificar.
 * @return true si la categoría existe, false en caso contrario.
 * @throws Exception Si ocurre un error en la ejecución SQL.
 */
public boolean existeCategoria(Connection conn, int idCategoria) throws Exception {
    String sql = "SELECT COUNT(*) FROM categorias WHERE id = ?";

    try (PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt(1, idCategoria);
        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return rs.getInt(1) > 0;
            }
        }
    }
    return false;
}
```

PedidoDAOImpl:

```java
public class PedidoDAOImpl implements GenericDAO<Pedido> {

    /**
     * Inserta un nuevo pedido en la base de datos.
     *
     * @param pedido Objeto Pedido con los datos a insertar.
     * @param conn Conexión activa a la base de datos.
     * @throws Exception Si ocurre un error durante la ejecución SQL.
     */
    @Override
    public void crear(Pedido pedido, Connection conn) throws Exception {
        String sql = "INSERT INTO pedidos (fecha, total) VALUES (?, ?)";

        try (PreparedStatement stmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
            stmt.setDate(1, new java.sql.Date(pedido.getFecha().getTime()));
            stmt.setDouble(2, pedido.getTotal());

            stmt.executeUpdate();

            try (ResultSet rs = stmt.getGeneratedKeys()) {
                if (rs.next()) {
                    pedido.setId(rs.getInt(1));
                }
            }
        }
    }
```

```java
    /**
     * Obtiene un pedido por su ID.
     *
     * @param id Identificador único del pedido.
     * @param conn Conexión activa a la base de datos.
     * @return Pedido encontrado o null si no existe.
     * @throws Exception Si ocurre un error durante la ejecución SQL.
     */
    @Override
    public Pedido leer(int id, Connection conn) throws Exception {
        String sql = "SELECT * FROM pedidos WHERE id = ?";

        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            stmt.setInt(1, id);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return new Pedido(
                            rs.getInt("id"),
                            rs.getDate("fecha"),
                            rs.getDouble("total")
                    );
                }
            }
        }
        return null;
    }
```

```java
/**
 * Lista todos los pedidos almacenados en la base de datos.
 *
 * @param conn Conexión activa a la base de datos.
 * @return Lista de pedidos.
 * @throws Exception Si ocurre un error durante la ejecución SQL.
 */
@Override
public List<Pedido> listar(Connection conn) throws Exception {
    String sql = "SELECT * FROM pedidos";
    List<Pedido> pedidos = new ArrayList<>();

    try (PreparedStatement stmt = conn.prepareStatement(sql); ResultSet rs = stmt.executeQuery()) {
        while (rs.next()) {
            pedidos.add(new Pedido(
                    rs.getInt("id"),
                    rs.getDate("fecha"),
                    rs.getDouble("total")
            ));
        }
    }
    return pedidos;
}
```

```java
/**
 * Actualiza los datos de un pedido existente.
 *
 * @param pedido Pedido con los datos actualizados (debe incluir el id).
 * @param conn Conexión activa a la base de datos.
 * @throws Exception Si ocurre un error durante la ejecución SQL.
 */
@Override
public void actualizar(Pedido pedido, Connection conn) throws Exception {
    String sql = "UPDATE pedidos SET fecha = ?, total = ? WHERE id = ?";

    try (PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setDate(1, new java.sql.Date(pedido.getFecha().getTime()));
        stmt.setDouble(2, pedido.getTotal());
        stmt.setInt(3, pedido.getId());

        stmt.executeUpdate();
    }
}

/**
 * Elimina un pedido de la base de datos por su ID.
 *
 * @param id Identificador del pedido a eliminar.
 * @param conn Conexión activa a la base de datos.
 * @throws Exception Si ocurre un error durante la ejecución SQL.
 */
@Override
public void eliminar(int id, Connection conn) throws Exception {
    String sql = "DELETE FROM pedidos WHERE id = ?";

    try (PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt(1, id);
        stmt.executeUpdate();
    }
}
```

```java
/**
 * Muestra el detalle de un pedido por consola, incluyendo producto,
 * categoria, cantidad y subtotal.
 *
 * @param conn Conexión activa a la base de datos.
 * @param pedidoId ID del pedido cuyo detalle se quiere mostrar.
 * @throws SQLException Si ocurre un error durante la ejecución SQL.
 */
public void mostrarDetallePedido(Connection conn, int pedidoId) throws SQLException {
    String sql = """
    SELECT
        p.nombre AS producto,
        c.nombre AS categoria,
        ip.cantidad,
        ip.subtotal
    FROM pedidos pe
    JOIN items_pedido ip ON pe.id = ip.pedido_id
    JOIN productos p ON ip.producto_id = p.id
    JOIN categorias c ON p.id_categoria = c.id
    WHERE pe.id = ?
""";

    try (PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt(1, pedidoId);
        try (ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                String producto = rs.getString("producto");
                String categoria = rs.getString("categoria");
                int cantidad = rs.getInt("cantidad");
                double subtotal = rs.getDouble("subtotal");

                System.out.println("Producto=" + producto
                        + ", Categoria=" + categoria
                        + ", Cantidad=" + cantidad
                        + ", Subtotal=" + subtotal);
            }
        }
    }
}
```

ItemPedidoDAOImpl:

```java
public class ItemPedidoDAOImpl implements GenericDAO<ItemPedido> {

    /**
     * Inserta un nuevo item de pedido en la base de datos.
     *
     * @param item Objeto ItemPedido con los datos a insertar.
     * @param conn Conexión activa a la base de datos.
     * @throws Exception Si ocurre un error durante la ejecución SQL.
     */
    @Override
    public void crear(ItemPedido item, Connection conn) throws Exception {
        String sql = "INSERT INTO items_pedido (pedido_id, producto_id, cantidad, subtotal) VALUES (?, ?, ?, ?)";
        try (PreparedStatement stmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
            stmt.setInt(1, item.getIdPedido());
            stmt.setInt(2, item.getIdProducto());
            stmt.setInt(3, item.getCantidad());
            stmt.setDouble(4, item.getSubtotal());
            stmt.executeUpdate();

            try (ResultSet rs = stmt.getGeneratedKeys()) {
                if (rs.next()) {
                    item.setId(rs.getInt(1));
                }
            }
        }
    }
```

```java
/**
 * Obtiene un item de pedido por su ID.
 *
 * @param id Identificador único del item de pedido.
 * @param conn Conexión activa a la base de datos.
 * @return ItemPedido encontrado o null si no existe.
 * @throws Exception Si ocurre un error durante la ejecución SQL.
 */
@Override
public ItemPedido leer(int id, Connection conn) throws Exception {
    String sql = "SELECT * FROM items_pedido WHERE id = ?";
    try (PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt(1, id);
        try (ResultSet rs = stmt.executeQuery()) {
            if (rs.next()) {
                return new ItemPedido(
                        rs.getInt("id"),
                        rs.getInt("id_pedido"),
                        rs.getInt("id_producto"),
                        rs.getInt("cantidad"),
                        rs.getDouble("subtotal")
                );
            }
        }
    }
    return null;
}
```

```java
/**
 * Lista todos los items de pedido existentes en la base de datos.
 *
 * @param conn Conexión activa a la base de datos.
 * @return Lista de ItemPedido.
 * @throws Exception Si ocurre un error durante la ejecución SQL.
 */
@Override
public List<ItemPedido> listar(Connection conn) throws Exception {
    List<ItemPedido> lista = new ArrayList<>();
    String sql = "SELECT * FROM item_pedido";
    try (PreparedStatement stmt = conn.prepareStatement(sql); ResultSet rs = stmt.executeQuery()) {
        while (rs.next()) {
            lista.add(new ItemPedido(
                    rs.getInt("id"),
                    rs.getInt("id_pedido"),
                    rs.getInt("id_producto"),
                    rs.getInt("cantidad"),
                    rs.getDouble("subtotal")
            ));
        }
    }
    return lista;
}
```

```java
/**
 * Actualiza los datos de un item de pedido existente.
 *
 * @param item ItemPedido con los datos actualizados (debe incluir el id).
 * @param conn Conexión activa a la base de datos.
 * @throws Exception Si ocurre un error durante la ejecución SQL.
 */
@Override
public void actualizar(ItemPedido item, Connection conn) throws Exception {
    String sql = "UPDATE items_pedido SET id_pedido = ?, id_producto = ?, cantidad = ?, subtotal = ? WHERE id = ?";
    try (PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt(1, item.getIdPedido());
        stmt.setInt(2, item.getIdProducto());
        stmt.setInt(3, item.getCantidad());
        stmt.setDouble(4, item.getSubtotal());
        stmt.setInt(5, item.getId());
        stmt.executeUpdate();
    }
}

/**
 * Elimina un item de pedido de la base de datos por su ID.
 *
 * @param id Identificador del item de pedido a eliminar.
 * @param conn Conexión activa a la base de datos.
 * @throws Exception Si ocurre un error durante la ejecución SQL.
 */
@Override
public void eliminar(int id, Connection conn) throws Exception {
    String sql = "DELETE FROM items_pedido WHERE id = ?";
    try (PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt(1, id);
        stmt.executeUpdate();
    }
}
```

Service:

CategoriaServiceImpl:

```java
public class CategoriaServiceImpl {

    private final CategoriaDAOImpl categoriaDAO;

    public CategoriaServiceImpl(CategoriaDAOImpl categoriaDAO) {
        this.categoriaDAO = categoriaDAO;
    }

    /**
     * Crea una nueva categoría en la base de datos. Valida que el nombre no sea
     * nulo ni vacío y que no exista ya una categoría con el mismo nombre.
     *
     * @param categoria Objeto Categoria a crear.
     * @throws IllegalArgumentException Si el nombre es nulo, vacío o ya existe
     * una categoría con ese nombre.
     * @throws Exception Si ocurre algún error durante la operación en la base
     * de datos.
     */
    public void crear(Categoria categoria) throws Exception {
        if (categoria.getNombre() == null || categoria.getNombre().trim().isEmpty()) {
            throw new IllegalArgumentException("El nombre de la categoría no puede estar vacío.");
        }

        Connection conn = null;
        try {
            conn = DatabaseConnection.getConnection();
            conn.setAutoCommit(false);

            if (categoriaDAO.existeNombre(categoria.getNombre(), conn)) {
                throw new IllegalArgumentException("Ya existe una categoría con ese nombre.");
            }

            categoriaDAO.crear(categoria, conn);
            conn.commit();
        } catch (Exception ex) {
            if (conn != null) {
                conn.rollback();
            }
            throw ex;
        } finally {
            if (conn != null) {
                conn.close();
            }
        }
    }
```

```java
/**
 * Actualiza una categoría existente en la base de datos.
 *
 * @param categoria Objeto Categoría con datos actualizados (debe incluir el
 * id).
 * @throws Exception Si ocurre algún error durante la operación en la base
 * de datos.
 */
public void actualizar(Categoria categoria) throws Exception {
    Connection conn = null;
    try {
        conn = DatabaseConnection.getConnection();
        conn.setAutoCommit(false);

        categoriaDAO.actualizar(categoria, conn);

        conn.commit();
    } catch (Exception ex) {
        if (conn != null) {
            conn.rollback();
        }
        throw ex;
    } finally {
        if (conn != null) {
            conn.close();
        }
    }
}
```

```java
/**
 * Elimina una categoría de la base de datos según su ID.
 *
 * @param id Identificador de la categoría a eliminar.
 * @throws Exception Si ocurre algún error durante la operación en la base
 * de datos.
 */
public void eliminar(int id) throws Exception {
    Connection conn = null;
    try {
        conn = DatabaseConnection.getConnection();
        conn.setAutoCommit(false);

        categoriaDAO.eliminar(id, conn);

        conn.commit();
    } catch (Exception ex) {
        if (conn != null) {
            conn.rollback();
        }
        throw ex;
    } finally {
        if (conn != null) {
            conn.close();
        }
    }
}

/**
 * Obtiene una categoría por su ID.
 *
 * @param id Identificador de la categoría.
 * @return Objeto Categoría o null si no existe.
 * @throws Exception Si ocurre algún error durante la operación en la base
 * de datos.
 */
public Categoria leer(int id) throws Exception {
    try (Connection conn = DatabaseConnection.getConnection()) {
        return categoriaDAO.leer(id, conn);
    }
}
```

```java
/**
 * Lista todas las categorías existentes en la base de datos.
 *
 * @return Lista de categorías.
 * @throws Exception Si ocurre algún error durante la operación en la base
 * de datos.
 */
public List<Categoria> listar() throws Exception {
    try (Connection conn = DatabaseConnection.getConnection()) {
        return categoriaDAO.listar(conn);
    }
}
```

ProductoServiceImpl:

```java
public class ProductoServiceImpl {

    private final ProductoDAOImpl productoDAO;

    public ProductoServiceImpl() {
        this.productoDAO = new ProductoDAOImpl();
    }

    /**
     * Valida los datos de un producto antes de insertarlo o actualizarlo.
     * Verifica que el nombre no sea vacío, el precio y la cantidad sean mayores
     * a cero, y que la categoría asociada exista si se especifica.
     *
     * @param p Producto a validar.
     * @param conn Conexión activa a la base de datos para validar existencia de
     * categoría.
     * @throws Exception Si alguna validación falla.
     */
    private void validar(Producto p, Connection conn) throws Exception {
        if (p.getNombre() == null || p.getNombre().trim().isEmpty()) {
            throw new Exception("El nombre no puede estar vacío.");
        }
        if (p.getPrecio() <= 0) {
            throw new Exception("El precio debe ser mayor a 0.");
        }
        if (p.getCantidad() <= 0) {
            throw new Exception("La cantidad debe ser mayor a 0.");
        }
        if (p.getIdCategoria() != 0) {
            if (!productoDAO.existeCategoria(conn, p.getIdCategoria())) {
                throw new Exception("La categoría no existe.");
            }
        }
    }
```

```java
/**
 * Crea un nuevo producto en la base de datos tras validar los datos.
 * Realiza la operación en una transacción.
 *
 * @param p Producto a crear.
 * @return El producto creado con su ID asignado.
 * @throws Exception Si ocurre un error en la validación o en la operación
 * de base de datos.
 */
public Producto crear(Producto p) throws Exception {

    Connection conn = null;

    try {
        conn = DatabaseConnection.getConnection();
        conn.setAutoCommit(false);

        validar(p, conn);
        productoDAO.crear(p, conn);
        conn.commit();
        return p;
    } catch (Exception e) {
        if (conn != null) {
            conn.rollback();
        }
        throw e;
    } finally {
        if (conn != null) {
            conn.setAutoCommit(true);
            conn.close();
        }
    }
}
```

```java
/**
 * Obtiene un producto por su ID.
 *
 * @param id Identificador del producto.
 * @return Producto encontrado o null si no existe.
 * @throws Exception Si ocurre un error en la operación de base de datos.
 */
public Producto leer(int id) throws Exception {
    try (Connection conn = DatabaseConnection.getConnection()) {
        return productoDAO.leer(id, conn);
    }
}

/**
 * Actualiza un producto existente tras validar sus datos. Realiza la
 * operación en una transacción.
 *
 * @param p Producto con los datos actualizados.
 * @return Producto actualizado.
 * @throws Exception Si ocurre un error en la validación o en la operación
 * de base de datos.
 */
public Producto actualizar(Producto p) throws Exception {

    Connection conn = null;
    try {
        conn = DatabaseConnection.getConnection();
        conn.setAutoCommit(false);

        validar(p, conn);
        productoDAO.actualizar(p, conn);

        conn.commit();
        return p;
    } catch (Exception e) {
        if (conn != null) {
            conn.rollback();
        }
        throw e;
    } finally {
        if (conn != null) {
            conn.setAutoCommit(true);
            conn.close();
        }
    }
}
```

```java
/**
 * Elimina un producto de la base de datos por su ID. Realiza la operación
 * en una transacción.
 *
 * @param id Identificador del producto a eliminar.
 * @throws Exception Si ocurre un error en la operación de base de datos.
 */
public void eliminar(int id) throws Exception {
    Connection conn = null;
    try {
        conn = DatabaseConnection.getConnection();
        conn.setAutoCommit(false);

        productoDAO.eliminar(id, conn);

        conn.commit();
    } catch (Exception e) {
        if (conn != null) {
            conn.rollback();
        }
        throw e;
    } finally {
        if (conn != null) {
            conn.setAutoCommit(true);
            conn.close();
        }
    }
}

/**
 * Lista todos los productos almacenados en la base de datos.
 *
 * @return Lista de productos.
 * @throws Exception Si ocurre un error en la operación de base de datos.
 */
public List<Producto> listar() throws Exception {
    try (Connection conn = DatabaseConnection.getConnection()) {
        return productoDAO.listar(conn);
    }
}
```

```java
/**
 * Lista los productos filtrados por una categoría específica.
 *
 * @param idCategoria ID de la categoría para filtrar productos.
 * @return Lista de productos que pertenecen a la categoría dada.
 * @throws Exception Si ocurre un error en la operación de base de datos.
 */
public List<Producto> listarPorCategoria(int idCategoria) throws Exception {
    try (Connection conn = DatabaseConnection.getConnection()) {
        return productoDAO.listarPorCategoria(conn, idCategoria);
    }
}
```

PedidoServiceImpl:

```java
public class PedidoServiceImpl {

    private final PedidoDAOImpl pedidoDAO;
    private final ItemPedidoDAOImpl itemPedidoDAO;
    private final ProductoDAOImpl productoDAO;

    public PedidoServiceImpl(PedidoDAOImpl pedidoDAO, ItemPedidoDAOImpl itemPedidoDAO, ProductoDAOImpl productoDAO) {
        this.pedidoDAO = pedidoDAO;
        this.itemPedidoDAO = itemPedidoDAO;
        this.productoDAO = productoDAO;
    }
```

```java
/**
 * Crea un nuevo pedido junto con sus items, validando stock y actualizando
 * cantidades. La operación se realiza en una transacción para asegurar la
 * consistencia.
 *
 * @param pedido Pedido a crear (el total será calculado e actualizado).
 * @param items Lista de items que pertenecen al pedido.
 * @throws Exception Si algún producto no existe, no tiene stock suficiente
 * o ocurre un error en la base de datos.
 */
public void crearPedido(Pedido pedido, List<ItemPedido> items) throws Exception {
    Connection conn = null;
    try {
        conn = config.DatabaseConnection.getConnection();
        conn.setAutoCommit(false);

        // Validar stock para todos los items
        for (ItemPedido item : items) {
            Producto producto = productoDAO.leer(item.getIdProducto(), conn);
            if (producto == null) {
                throw new RuntimeException("Producto no encontrado ID " + item.getIdProducto());
            }
            if (producto.getCantidad() < item.getCantidad()) {
                throw new RuntimeException("Stock insuficiente para producto: " + producto.getNombre());
            }
        }

        // Crear pedido (total inicial 0, luego actualizamos)
        pedidoDAO.crear(pedido, conn);

        double totalPedido = 0;
```

```java
/**
 * Muestra el detalle completo de un pedido por su ID, incluyendo
 * información general y detalle de items. Si el pedido no existe, imprime
 * un mensaje indicando que no fue encontrado.
 *
 * @param pedidoId ID del pedido a mostrar.
 * @throws Exception Si ocurre un error al acceder a la base de datos.
 */
public void mostrarDetallePedido(int pedidoId) throws Exception {
    try (Connection conn = config.DatabaseConnection.getConnection()) {
        Pedido pedido = pedidoDAO.leer(pedidoId, conn);
        if (pedido == null) {
            System.out.println("Pedido no encontrado con ID " + pedidoId);
            return;
        }
        System.out.println("Pedido ID: " + pedido.getId() + ", Fecha: " + pedido.getFecha() + ", Total: " + pedido.getTotal());
        pedidoDAO.mostrarDetallePedido(conn, pedidoId);
    }
}
```

```
        // Crear items, calcular subtotal y actualizar stock
        for (ItemPedido item : items) {
            Producto producto = productoDAO.leer(item.getIdProducto(), conn);

            item.setIdPedido(pedido.getId());
            item.setSubtotal(producto.getPrecio() * item.getCantidad());
            totalPedido += item.getSubtotal();

            itemPedidoDAO.crear(item, conn);

            // Actualizar stock producto
            producto.setCantidad(producto.getCantidad() - item.getCantidad());
            productoDAO.actualizar(producto, conn);
        }

        // Actualizar total del pedido
        pedido.setTotal(totalPedido);
        pedidoDAO.actualizar(pedido, conn);

        conn.commit();
    } catch (Exception e) {
        if (conn != null) {
            conn.rollback();
        }
        throw e;
    } finally {
        if (conn != null) {
            conn.close();
        }
    }
```

DatabaseConnection:

```
public class DatabaseConnection {

    private static final HikariConfig config = new HikariConfig();
    private static final HikariDataSource ds;

    static {
        config.setJdbcUrl("jdbc:mysql://localhost:3307/db"); // Seteamos la url de nuestra BD
        config.setUsername("root"); // Usuario BD
        config.setPassword(""); // Password BD
        config.setMaximumPoolSize(10);  // Maximo 10 conexiones
        ds = new HikariDataSource(config);  // Inicialización del datasource con la configuración anterior
    }

    /**
     * Obtiene una conexión activa del pool de conexiones.
     *
     * @return Conexión a la base de datos.
     * @throws SQLException Si ocurre un error al obtener la conexión.
     */
    public static Connection getConnection() throws SQLException {
        return ds.getConnection();
    }
}
```

Main:

```java
public class main {

    public static void main(String[] args) {
        try {

            CategoriaServiceImpl categoriaService = new CategoriaServiceImpl(new dao.CategoriaDAOImpl());
            ProductoServiceImpl productoService = new ProductoServiceImpl();
            PedidoServiceImpl pedidoService = new PedidoServiceImpl(
                    new dao.PedidoDAOImpl(),
                    new dao.ItemPedidoDAOImpl(),
                    new dao.ProductoDAOImpl()
            );

            // 1) Crear categoria
            Categoria categoria = new Categoria(0, "Electrónica", "Productos electrónicos");
            categoriaService.crear(categoria);
            System.out.println("Categoria creada con ID: " + categoria.getId());

            // 2) Crear productos asociados a la categoria
            Producto producto1 = new Producto("Smartphone", "Teléfono inteligente", 50000.0, 10, categoria.getId());
            Producto producto2 = new Producto("Auriculares", "Auriculares inalámbricos", 15000.0, 20, categoria.getId());

            producto1 = productoService.crear(producto1);
            producto2 = productoService.crear(producto2);

            System.out.println("Producto 1 creado con ID: " + producto1.getId());
            System.out.println("Producto 2 creado con ID: " + producto2.getId());

            // 3) Crear pedido con items
            Pedido pedido = new Pedido();
            pedido.setFecha(new java.util.Date());
            pedido.setTotal(0);

            List<ItemPedido> items = new ArrayList<>();
            items.add(new ItemPedido(0, 0, producto1.getId(), 2, 0)); // 2 smartphones
            items.add(new ItemPedido(0, 0, producto2.getId(), 3, 0)); // 3 auriculares

            pedidoService.crearPedido(pedido, items);
            System.out.println("Pedido creado con ID: " + pedido.getId());

            // 4) Mostrar detalle del pedido
            System.out.println("\nDetalle del pedido:");
            pedidoService.mostrarDetallePedido(pedido.getId());

        } catch (Exception e) {
            System.err.println("Error en la ejecución: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

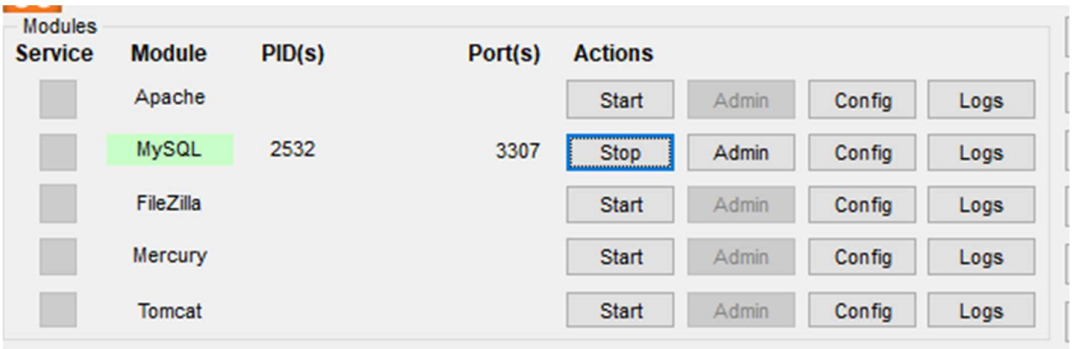
Salida por pantalla:

```
[main] INFO com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Starting...
[main] INFO com.zaxxer.hikari.pool.HikariPool - HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@6b1274d2
[main] INFO com.zaxxer.hikari.HikariDataSource - HikariPool-1 - Start completed.
Categor?a creada con ID: 6
Producto 1 creado con ID: 10
Producto 2 creado con ID: 11
Pedido creado con ID: 4

Detalle del pedido:
Pedido ID: 4, Fecha: 2025-08-11, Total: 145000.0
Producto=Smartphone, Categor?a=Electr?nica, Cantidad=2, Subtotal=100000.0
Producto=Auriculares, Categor?a=Electr?nica, Cantidad=3, Subtotal=45000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

RECORDATORIO:

Debemos tener nuestro puerto mysql activo a través de xampp:



Ademas nuestra BD debe estar conectada con nuestro puerto, lo podemos verificar en DBeaver: