UF6 - INTRODUCCIÓ A LA PERSISTÈNCIA EN LES BDS

M03 Programació

UF6 - Introducció a la persistència en les BDs

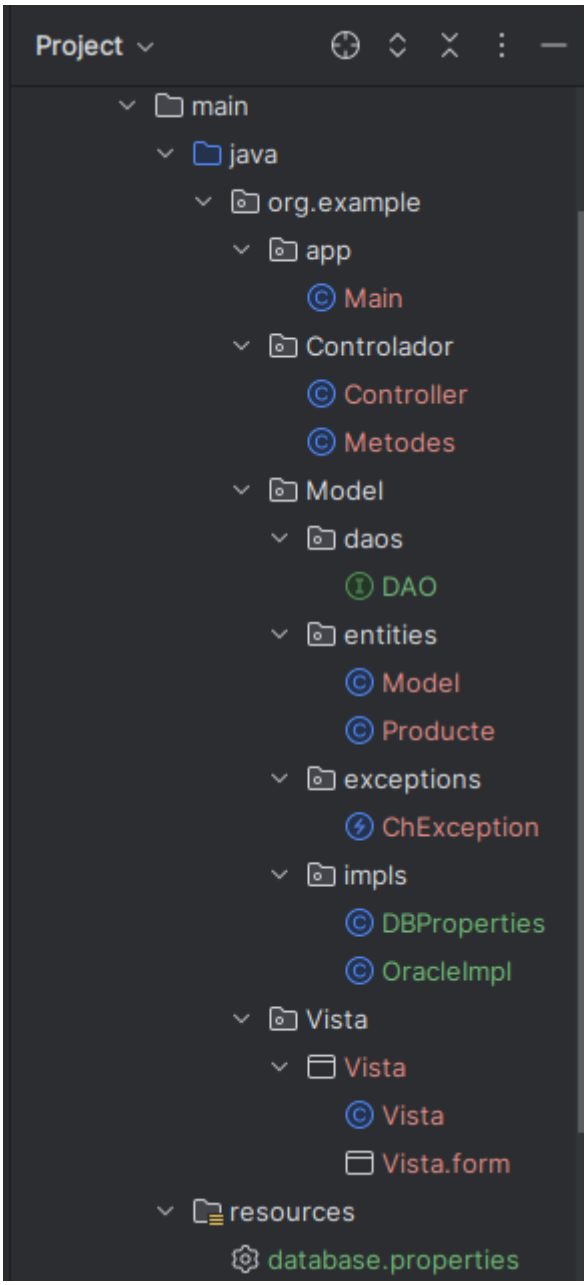
Christian Villanueva Lor

2023/2024

Índex:

Mode gràfic en patró MVC.

Al igual que en l'anterior projecte de l'UF5, en aquest projecte he seguit la mateixa metodologia per a l'estructura, la qual s'anomena MVC (Model Vista Controlador), on dins del paquet del Model, també podem trobar diferents paquets per a DAOs, entitats, excepcions o implementacions, mentre que en els paquets de Controlador i Vista, sols trobem els fitxers corresponents per a poder dur a terme la seva principal funcionalitat.



Connexió a una BD Oracle utilitzant l'API JDBC.

A l'hora d'intentar accedir a la base de dades, he decidit guardar les propietats com ara l'usuari, la contrasenya i l'URL per a poder accedir a la base de dades, en un fitxer d'extensió properties, el qual després, utilitzant la utilitat Properties, podem accedir a la seva informació de la següent forma.

```
public DBProperties() {
    try (InputStream input = getClass().getClassLoader().getResourceAsStream( name: "database.properties")) {
        if (input == null) {
            System.out.println("Lo siento, no se pudo encontrar el archivo database.properties");
            return;
        }
        // Cargar el archivo properties
        properties.load(input);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

Un cop està el fitxer carregat, podem accedir a cadascuna de les propietats mitjançant getters.

```
6 usages  new *
public String getUrl() { return properties.getProperty("db.url"); }

/**
 * Getter de l'usuari de la base de dades
 * @return usuari de la base de dades
 * */
6 usages  new *
public String getUsername() { return properties.getProperty("db.username"); }

/**
 * Getter de la contrasenya de la base de dades
 * @return contrasenya de la base de dades
 * */
6 usages  new *
public String getPassword() { return properties.getProperty("db.password"); }

/**
 * Getter del driver de la base de dades
 * @return driver de la base de dades
 * */
no usages  new *
public String getDriver() { return properties.getProperty("db.driver"); }
```

Després, per poder realitzar la connexió amb la base de dades, simplement haurem de crear un DBProperties, i ja es carregarà el fitxer i podrem accedir a les seves propietats.

```
18 usages
DBProperties connection = new DBProperties();

/**
 * Mètode que retorna tots els productes de la base de dades
 * @return Llista de productes
 * @throws ChException
 * */
1 usage  new *
@Override
public List<Producte> getAll() throws ChException {

    create();

    List<Producte> productes = new ArrayList<>();

    try (Connection con = DriverManager.getConnection(
            connection.getUrl(),
            connection.getUsername(),
            connection.getPassword()
    )) {
```

Interfície DAO

En la interfície DAO, he indicat els mètodes getAll, save, insert, modify, delete i create. Tots aquests mètodes han sigut implementats en la classe OracleImpl, ja que aquesta classe implementa la interfície DAO. Tots els mètodes utilitzen un prepared statement, i tots els mètodes també criden al mètode create, el qual comprova que la taula estigui creada, i en cas de no estar creada, la crea mitjançant un procés emmagatzemat anomenat creartaula. A continuació veurem com funciona cada mètode.

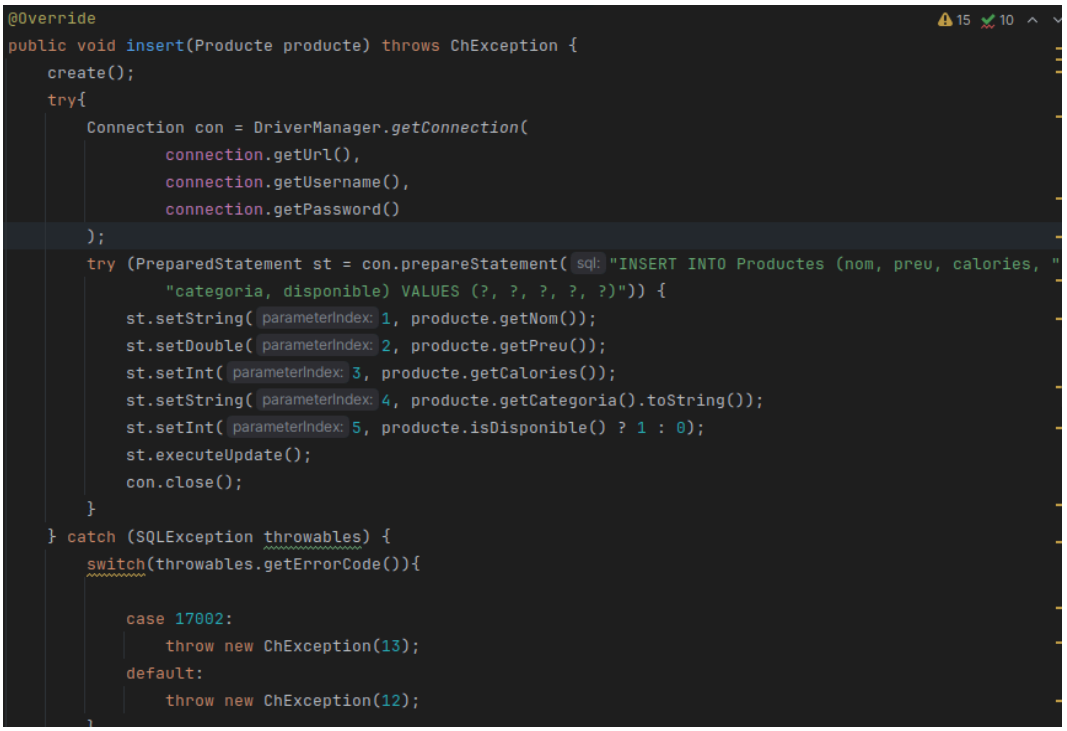
➢ getAll(): Realitza un prepareStatement com a consulta, i ho afegeix a una llista la qual retorna en acabar el procés.

```java
public List<Producte> getAll() throws ChException {
    create();
    List<Producte> productes = new ArrayList<>();
    try (Connection con = DriverManager.getConnection(
            connection.getUrl(),
            connection.getUsername(),
            connection.getPassword()
    )) {
        // Consultar los datos de la tabla
        try (PreparedStatement st = con.prepareStatement( sql: "SELECT * FROM Productes");
             ResultSet rs = st.executeQuery()) {

            while (rs.next()) {
                Producte producte = new Producte();
                producte.setNom(rs.getString( columnLabel: "nom"));
                producte.setPreu(rs.getDouble( columnLabel: "preu"));
                producte.setCalories(rs.getInt( columnLabel: "calories"));
                producte.setCategoriaStr(rs.getString( columnLabel: "categoria"));
                producte.setDisponible(rs.getInt( columnLabel: "disponible") == 1);
                productes.add(producte);
            }
            con.close();
        }
    } catch (SQLException throwables) {
        switch(throwables.getErrorCode()){
```

➢ save(): Aquest mètode realitza un commit de la connexió, per assegurar-se que tots els canvis es guarden.

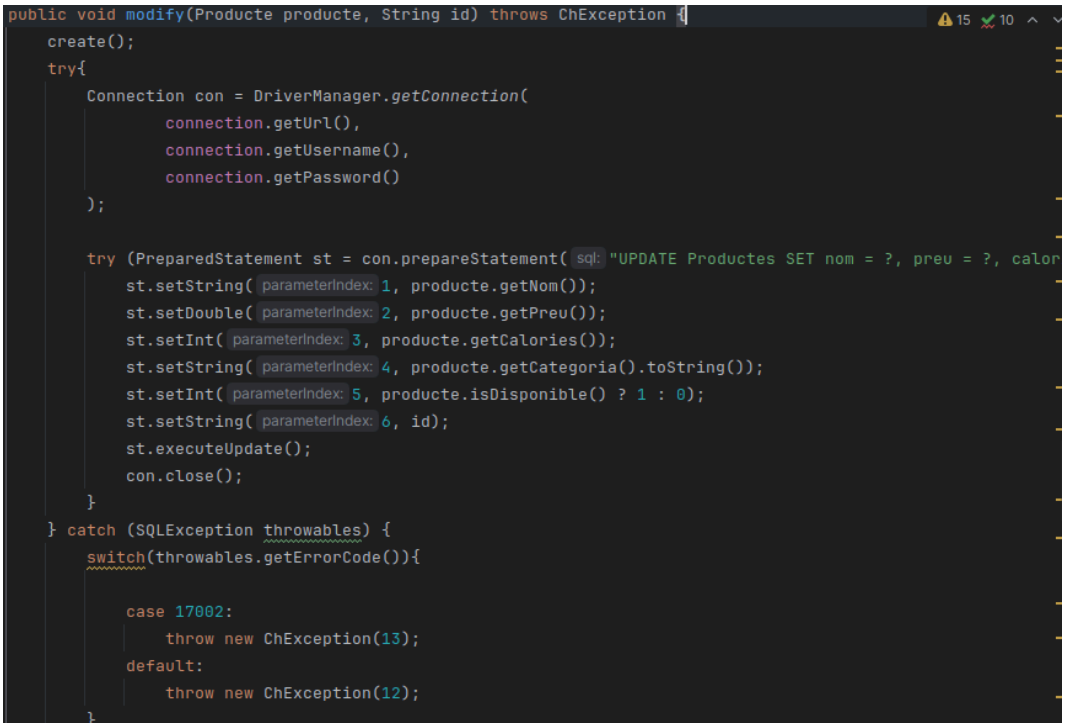
```java
1 usage  new *
@Override
public void save() throws ChException {
    try{
        Connection con = DriverManager.getConnection(
                connection.getUrl(),
                connection.getUsername(),
                connection.getPassword()
        );
        con.setAutoCommit(false);
        con.commit();
        con.close();
    } catch (SQLException throwables) {
        switch(throwables.getErrorCode()){

            case 17002:
                throw new ChException(13);
            default:
                throw new ChException(12);
        }
    }
}
```

➢ insert(): Aquest mètode, crida un prepareStatement, el qual mitjançant interrogants, i després amb setters i indicant la posició del interrogant corresponent per a aquell valor, podem inserir els valors en la base de dades perfectament.

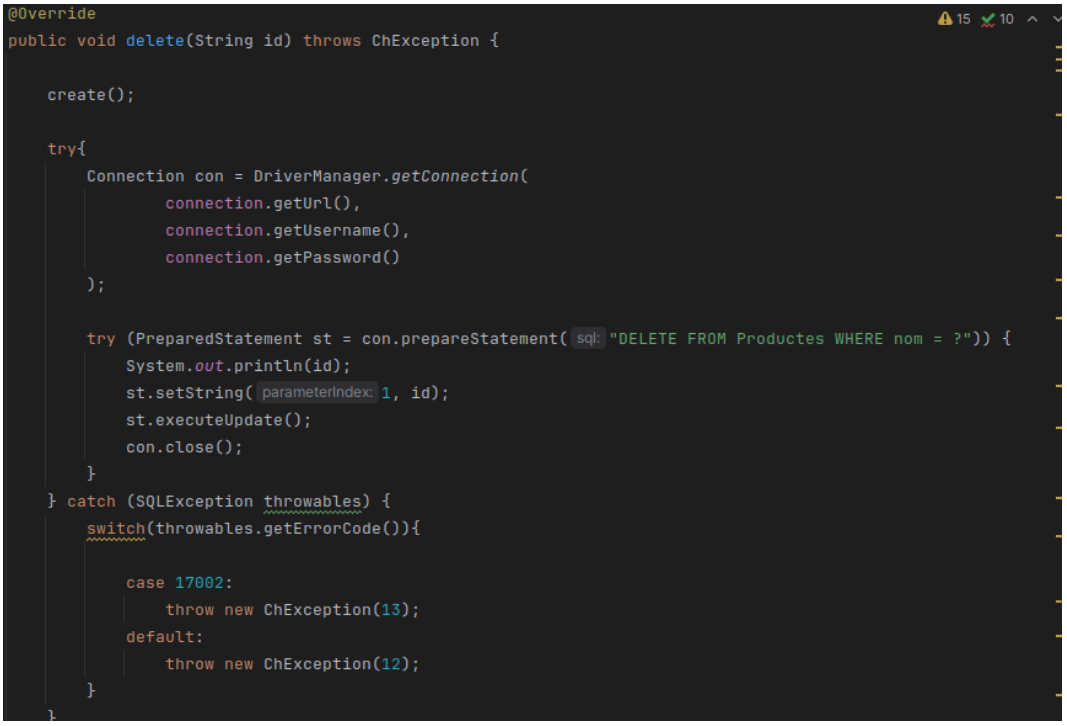


```
@Override                                                                    ⚠ 15  ✓ 10
public void insert(Producte producte) throws ChException {
    create();
    try{
        Connection con = DriverManager.getConnection(
                connection.getUrl(),
                connection.getUsername(),
                connection.getPassword()
        );
        try (PreparedStatement st = con.prepareStatement( sql: "INSERT INTO Productes (nom, preu, calories, "
                "categoria, disponible) VALUES (?, ?, ?, ?, ?)")) {
            st.setString( parameterIndex: 1, producte.getNom());
            st.setDouble( parameterIndex: 2, producte.getPreu());
            st.setInt( parameterIndex: 3, producte.getCalories());
            st.setString( parameterIndex: 4, producte.getCategoria().toString());
            st.setInt( parameterIndex: 5, producte.isDisponible() ? 1 : 0);
            st.executeUpdate();
            con.close();
        }
    } catch (SQLException throwables) {
        switch(throwables.getErrorCode()){

            case 17002:
                throw new ChException(13);
            default:
                throw new ChException(12);
```

➢ modify(): Aquest mètode funciona de la mateixa forma que el mètode d'inserir, però realitza un UPDATE en lloc d'un INSERT INTO, i també se l'indica l'ID del producte, el qual com pot ser també modificat, s'agafa el valor que havia abans de ser modificat.



➢ delete(): Aquest mètode funciona igual que els mètodes per inserir i modificar, però simplement fent un DELETE FROM, i indicant l'ID del producte a eliminar.



```
@Override
public void delete(String id) throws ChException {

    create();

    try{
        Connection con = DriverManager.getConnection(
                connection.getUrl(),
                connection.getUsername(),
                connection.getPassword()
        );

        try (PreparedStatement st = con.prepareStatement( sql: "DELETE FROM Productes WHERE nom = ?")) {
            System.out.println(id);
            st.setString( parameterIndex: 1, id);
            st.executeUpdate();
            con.close();
        }
    } catch (SQLException throwables) {
        switch(throwables.getErrorCode()){

            case 17002:
                throw new ChException(13);
            default:
                throw new ChException(12);
        }
    }
```

➢ create(): Aquest mètode fa una comprovació per saber si la taula Productes existeix en la base de dades, i en cas que no sigui així, executa un procés emmagatzemat anomenat creartaula, el qual crea la taula amb tots els camps necessaris.

```
@Override                                                                    15
public void create() throws ChException{
    try (Connection con = DriverManager.getConnection(
            connection.getUrl(),
            connection.getUsername(),
            connection.getPassword()
    )) {
        DatabaseMetaData dbm = con.getMetaData();
        try (ResultSet tables = dbm.getTables( catalog: null,  schemaPattern: null,
                tableNamePattern: "PRODUCTES",  types: null)) {
            if (!tables.next()) {
                try (PreparedStatement st = con.prepareStatement( sql: "BEGIN creartaula; END;")) {
                    st.executeUpdate();
                }
            }
        }
    } catch (SQLException throwables) {
        switch(throwables.getErrorCode()){

            case 17002:
                throw new ChException(13);
            default:
                throw new ChException(12);
        }
    }
}
```

Expressions regulars

En aquest projecte, he utilitzat dues expressions regulars, per verificar que el nom del producte comenci en majúscula, i no tingui cap signe de puntuació en tot el nom.

```
if (Pattern.compile( regex: "^[^A-Z]").matcher(String.valueOf(view.getCampNom().getText().charAt(0))).find()) {
    throw new ChException(8);
}

if (Pattern.compile( regex: "\\p{Punct}").matcher(String.valueOf(view.getCampNom().getText())).find()) {
    throw new ChException(8);
}
```

Enumeracions dins un JComboBox

En aquest projecte, he fet que la categoria del producte, en lloc de ser un simple String, sigui una categoria definida en una Enumeració.

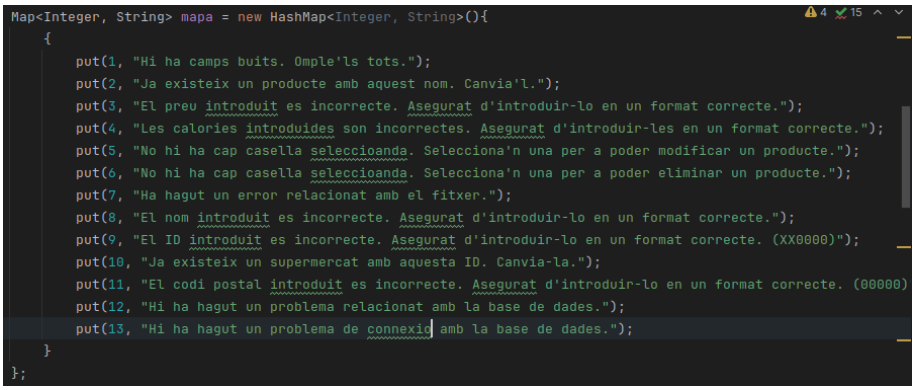
```
public enum Categoria {
    1 usage
    Fresc,
    1 usage
    Processat,
    1 usage
    Cereal,
    1 usage
    Beguda,
    1 usage
    Congelat,
}
```

De manera que perquè les categories apareguin en el JComboBOX, he fet un bucle que recorre tots els valors de l'enumeració, i afegeix cada valor al JComboBox.

```
categoriaSelector.addItem("Selecciona una categoria");

for (int i = 0; i < Producte.Categoria.values().length; i++) {
    categoriaSelector.addItem(Producte.Categoria.values()[i].toString());
}
```

Excepcions

En el cas de les excepcions, he deixat igual que en l'anterior projecte, però he afegit dues excepcions més, les quals són relacionades en la base de dades, i una d'elles serveix per indicar que la connexió amb la base de dades no ha tingut èxit, i l'altra per indicar que ha sorgit un problema desconegut.



Després, per tractar les excepcions, ho he fet amb el propertyChange, igual que en l'anterior projecte, i indicant cada excepció amb el mètode setExcepcio.