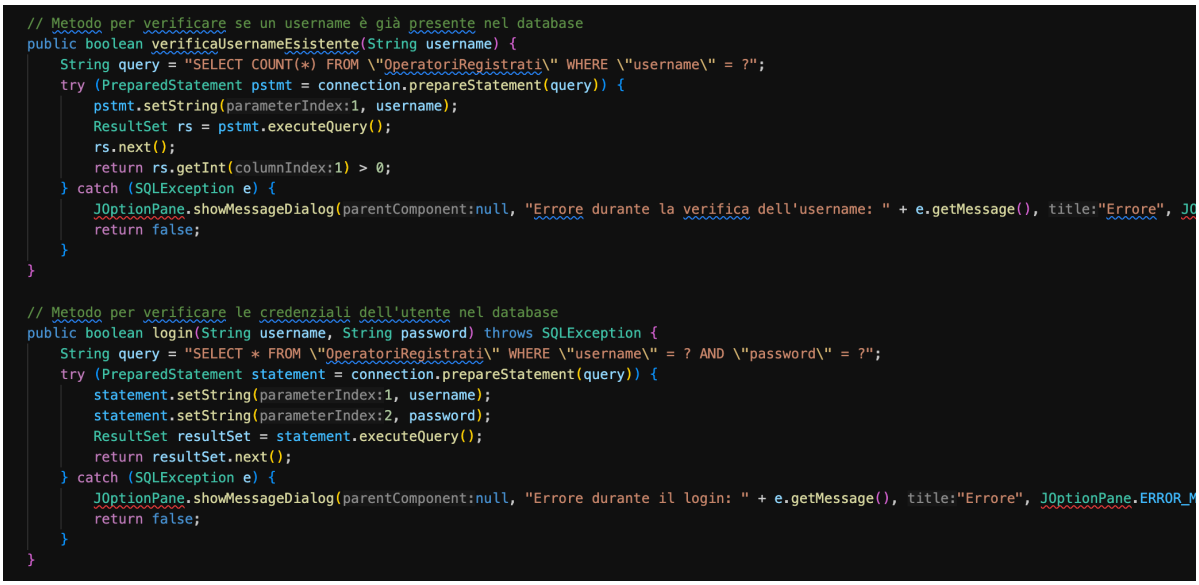
DatabaseManager

1. Singleton Pattern

- **Dove si applica:** nella gestione della connessione al database nella classe DatabaseManager
- **Motivazione:** garantire che la classe DatabaseManager abbia una singola istanza che gestisce la connessione al database. Questo è utile per evitare la creazione di più connessioni al database, che potrebbero causare problemi di prestazioni e di concorrenza. Il Singleton Pattern è usato per garantire che ci sia una sola istanza di connessione al database condivisa in tutta l'applicazione

2. Factory Method Pattern

- **Dove si applica:** nella creazione della connessione al database all'interno di DatabaseManager
- **Motivazione:** è usato per incapsulare la logica di creazione della connessione al database, permettendo di fornire parametri diversi come URL, username e password. Questo rende il processo di creazione della connessione flessibile e estendibile
- **Codice rilevante**



```java
// Metodo per verificare se un username è già presente nel database
public boolean verificaUsernameEsistente(String username) {
    String query = "SELECT COUNT(*) FROM \"OperatoriRegistrati\" WHERE \"username\" = ?";
    try (PreparedStatement pstmt = connection.prepareStatement(query)) {
        pstmt.setString(parameterIndex:1, username);
        ResultSet rs = pstmt.executeQuery();
        rs.next();
        return rs.getInt(columnIndex:1) > 0;
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(parentComponent:null, "Errore durante la verifica dell'username: " + e.getMessage(), title:"Errore", JO
        return false;
    }
}

// Metodo per verificare le credenziali dell'utente nel database
public boolean login(String username, String password) throws SQLException {
    String query = "SELECT * FROM \"OperatoriRegistrati\" WHERE \"username\" = ? AND \"password\" = ?";
    try (PreparedStatement statement = connection.prepareStatement(query)) {
        statement.setString(parameterIndex:1, username);
        statement.setString(parameterIndex:2, password);
        ResultSet resultSet = statement.executeQuery();
        return resultSet.next();
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(parentComponent:null, "Errore durante il login: " + e.getMessage(), title:"Errore", JOptionPane.ERROR_M
        return false;
    }
}
```

3. Strategy Pattern

- **Dove si applica:** per la gestione di connessioni diverse o operazioni diverse come verificaUsernameEsistente, login, inserisci dati
- **Motivazione:** necessità di variare il comportamento di alcuni metodi come login o verificaUsernameEsistente senza modificare il codice della classe principale. Questo pattern permette di definire una famiglia di algoritmi, incapsularli e renderli intercambiabili