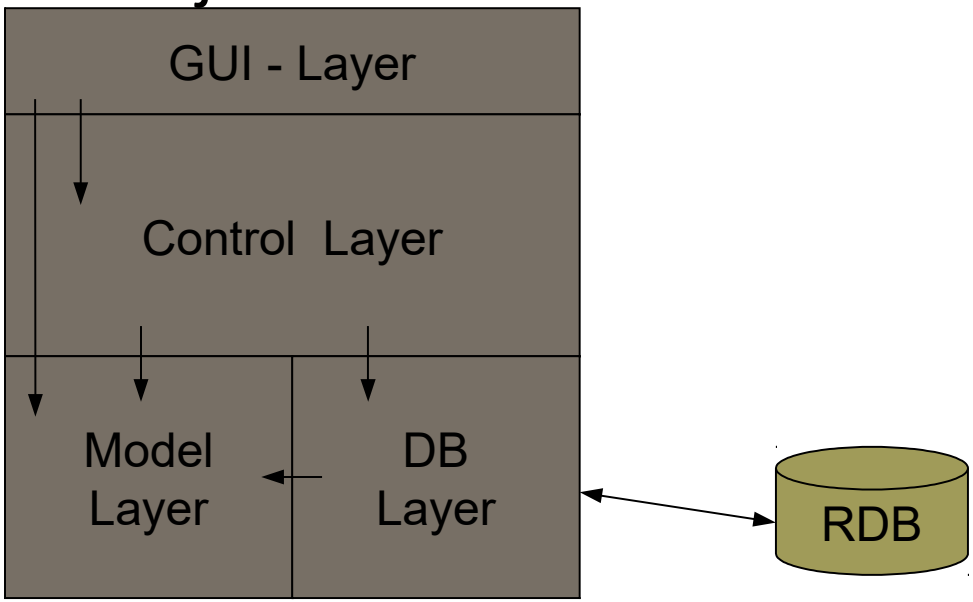
Architecture

Layered Architecture, Data Access Objects



Database Access from Java

- Database Access uses the package java.sql:
  - `import java.sql.*;`
- We use the classes below:
  - *DriverManager*
  - *Connection*
  - *Statement*
  - *ResultSet*
  - *DatabaseMetaData*
  - *ResultSetMetaData*

The Connection Class

- The purpose of this class is to handle the database connection.
- Is implemented using the *Singleton Pattern*
  - The singleton pattern ensures that only one instance of a class is created.
  - All objects that use a database connection use the same instance of the connection.
- Methods to handle transaction
  - start transaction
  - end transaction

Transactions in SQL

- A **single SQL-statement** is treated as an **transaction**, that is executed with the **ACID** properties

  (Also an update like:

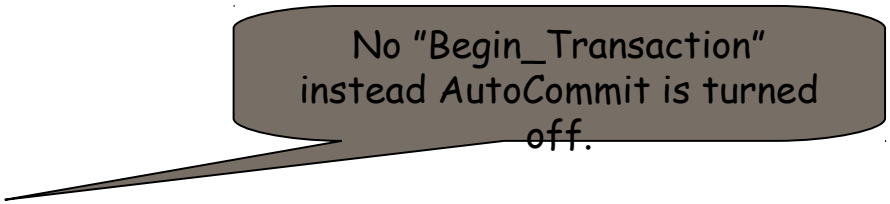
  ```
  UPDATE    EMPLOYEE
  SET       SALARY = SALARY*1.1
  WHERE SSN = '123456789'
  ```

  which actually consists of both a read and a write.)

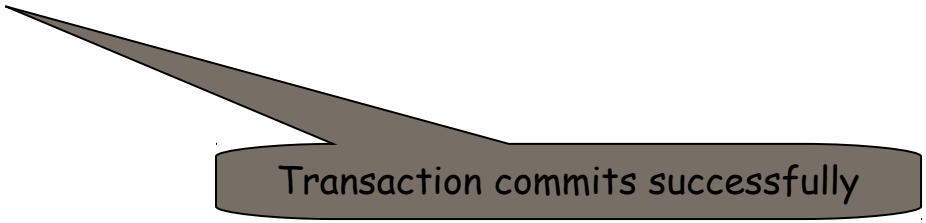
- Embedded SQL provides statements to handle transactions, typically statements like:
  - BEGIN_TRANSCATION
  - COMMIT
  - ROLLBACK

  are offered (for JDBC, see next slide).

Transactions in JDBC/SQL




```
con.setAutoCommit(false);
Statement stmt = con.createStatement();
stmt.executeUpdate( "INSERT INTO Materiale VALUES(.. . .)");
stmt.executeUpdate( "INSERT INTO CD  VALUES(.. . .)");
con.commit();
con.setAutoCommit(true);
```



DBConnection

```
public static void startTransaction(){
    try{
        con.setAutoCommit(false);
    }
    catch(Exception e){
        // ...
    }
}
```

Committing a transaction

```
public staic void commitTransaction(){
    try{
        con.commit();
    }
    catch(Exception e){
        con.rollback();
    }
    finally {
       con.setAutoCommit(true);
    }
}
```

DBConnection

```
public static void rollbackTransaction() {
    try{
        con.rollback();
    }
    catch(Exception e){
    }
    finally {
        con.setAutoCommit(true);
    }
}
```

Transaction I controller

```
public void insertNew(String fname, String lname,
                      double salary, String superssn){

    Employee empObj = new Employee();
    empObj.setFname(fname);
    empObj.setLname(lname);
    empObj.setSupervisor(new Employee(superssn));
    try{
        DbConnection.startTransaction();
        DBEmployee dbEmp = new DBEmployee();
        dbEmp.insertEmployee(empObj);
        DbConnection.commitTransaction();
    }
    catch(Exception e){
        DbConnection.rollbackTransaction();
    }
}
```

BeginTransaction

Transaction commits successfully

Insert is aborted. The entire transaction is rolled back

Error handling

- Where is the error detected?
- Who will take care of the error handling?
  - DB-layer: Detect, fix if possible
  - Controller: Optionally convert to custom Exception
  - UI-layer: Let user handle exceptions that we can't solve in code.
    - JOptionPane.showMessageDialog(...)