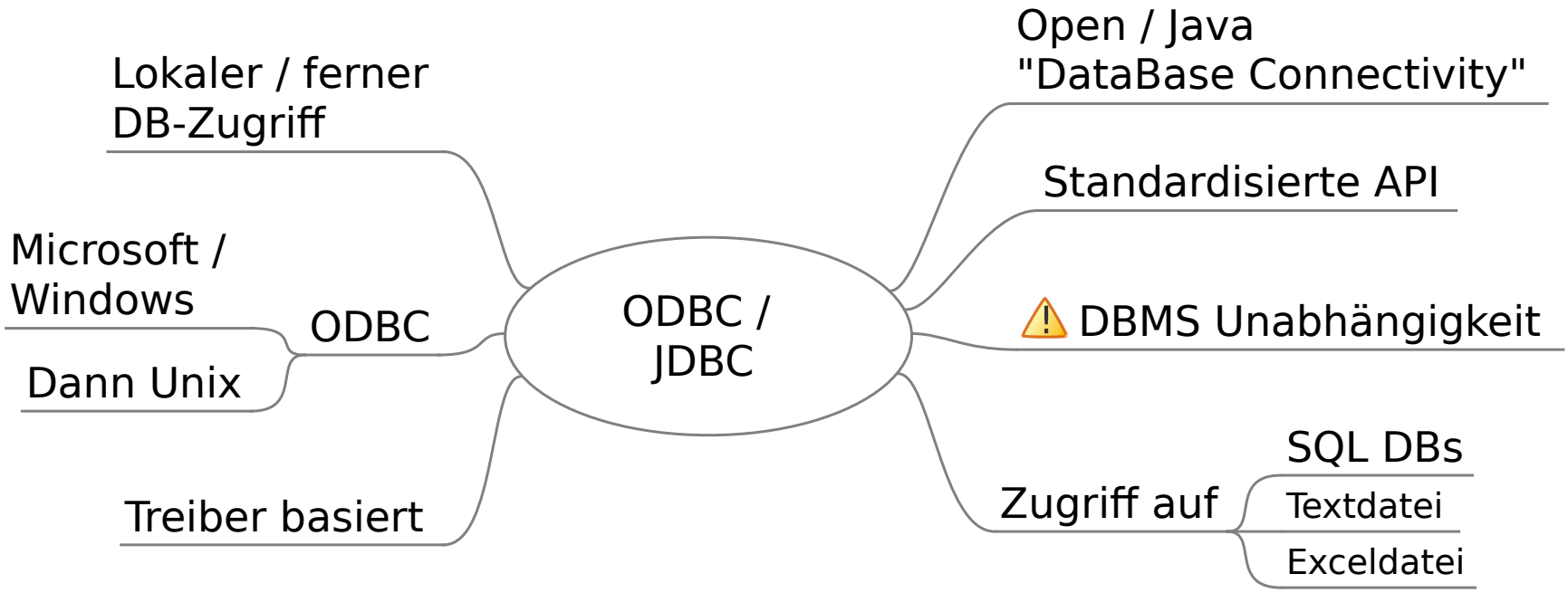
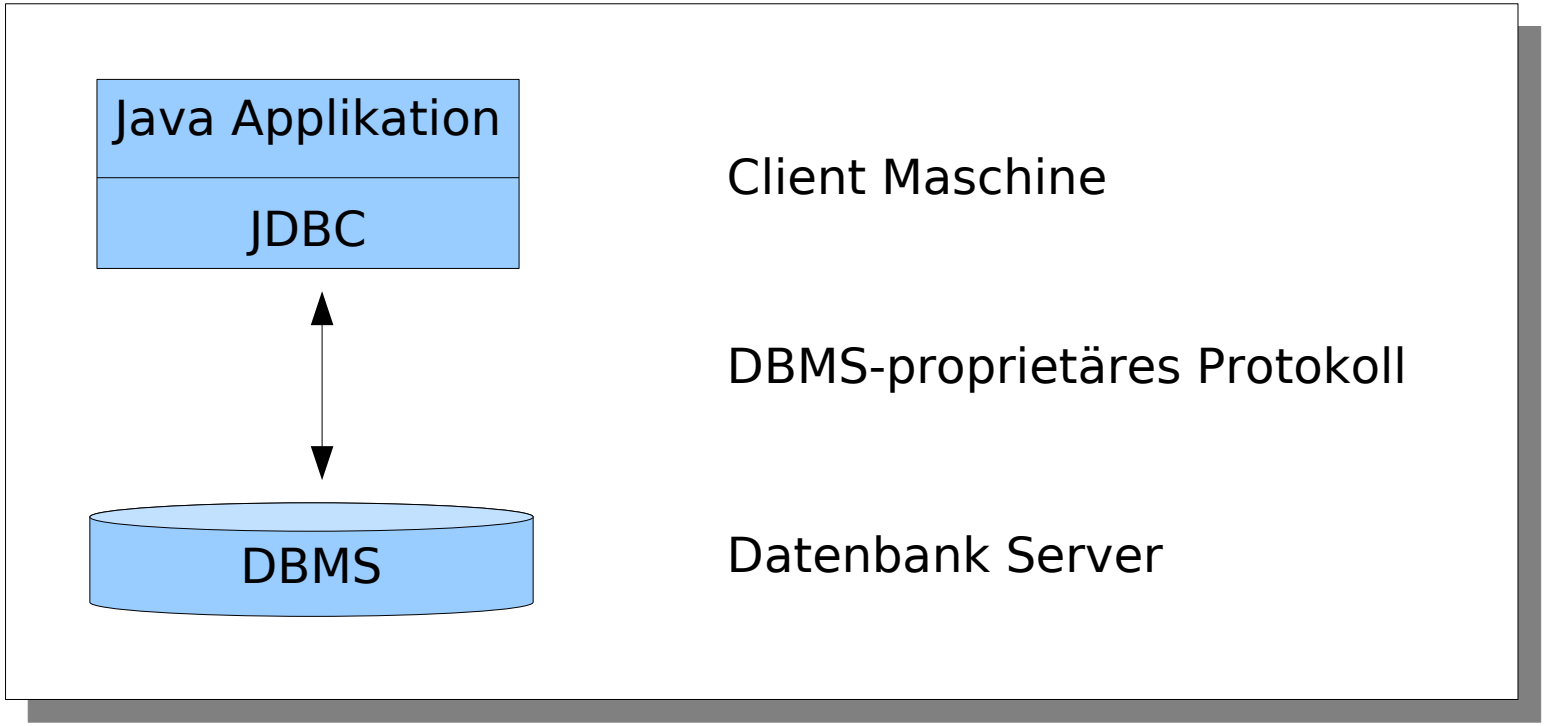
Tag 5
Inhaltsverzeichnis

ODBC / JDBC
Prinzipien

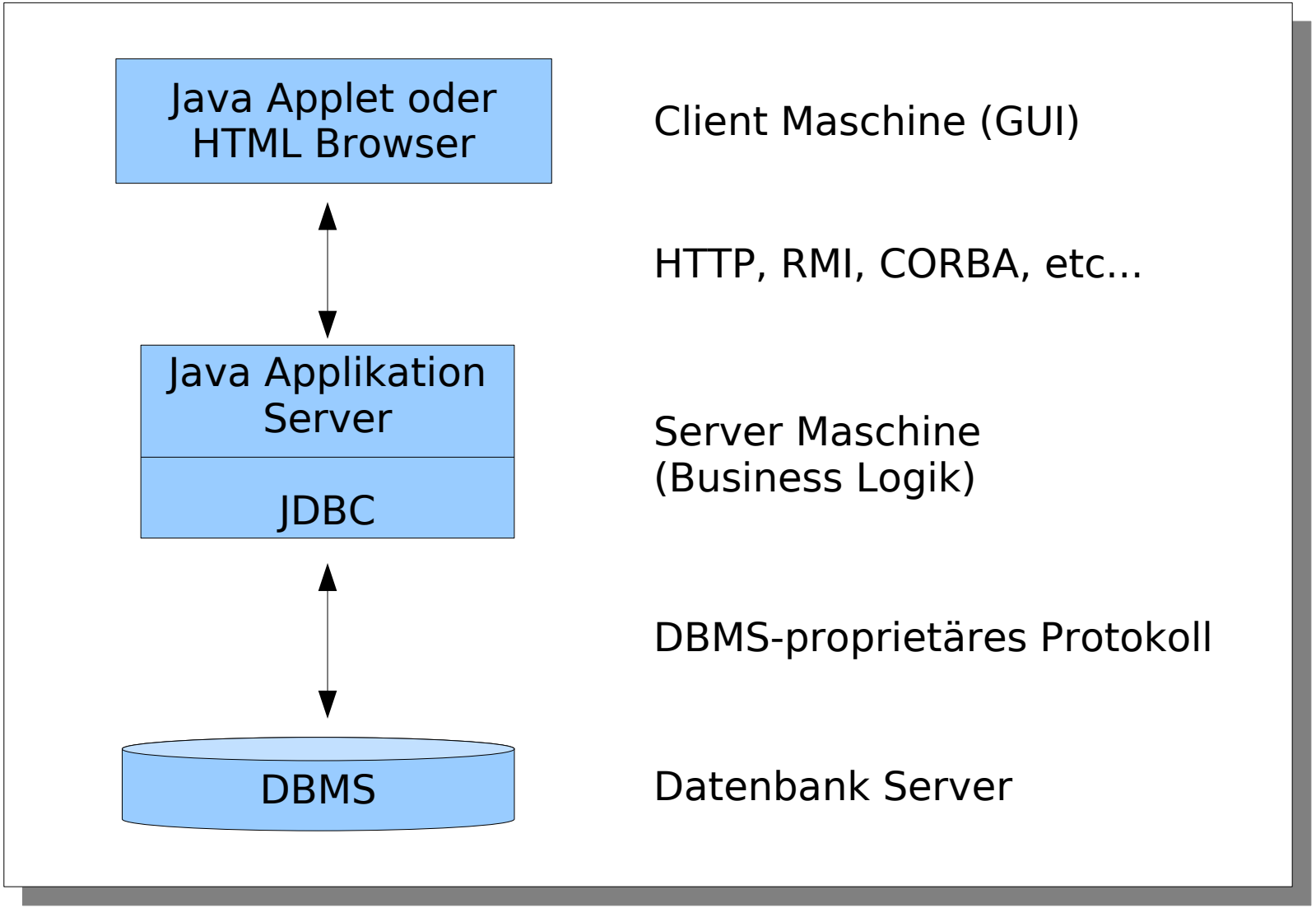


Quelle: https://de.wikipedia.org/wiki/Java_Database_Connectivity

JDBC
"Two-tier" Architektur für Datenzugriff



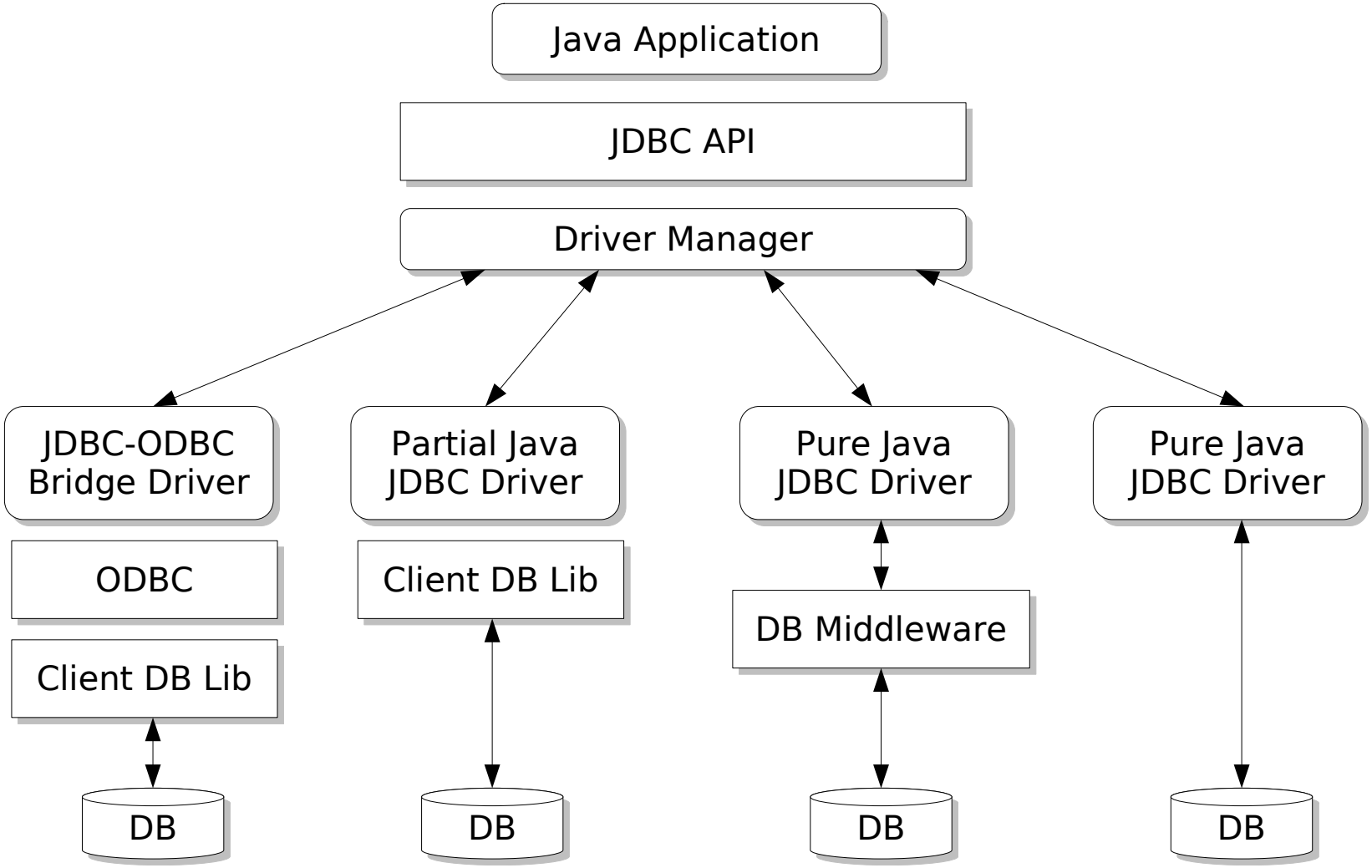
JDBC
"Three-tier" Architektur für Datenzugriff



JDBC
Komponenten

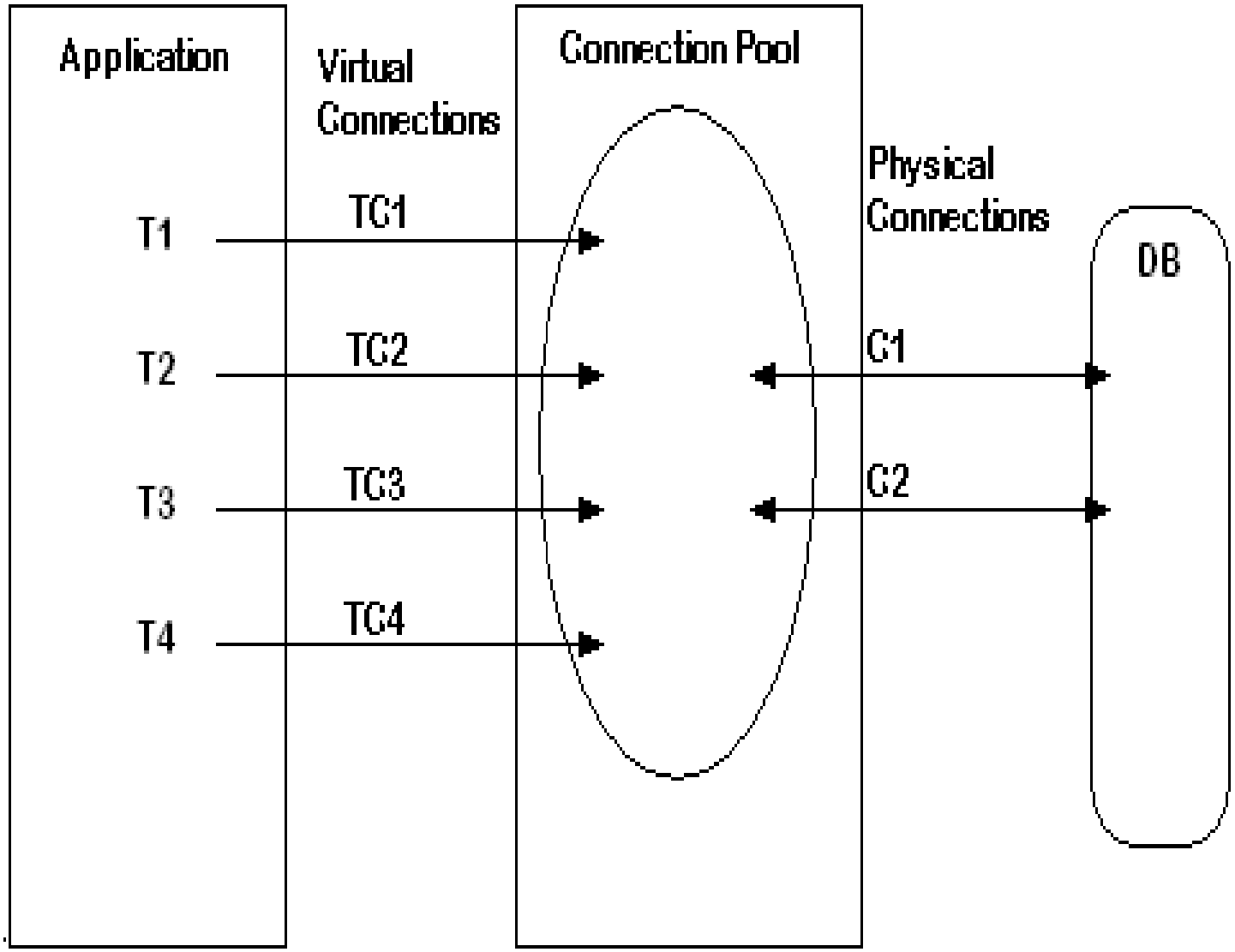
- JDBC API
  - Teil von JavaSE & JavaEE
- JDBC Driver Manager
  - DriverManager
  - DataSource + Java Naming and Directory Interface (JNDI)
- JDBC Test Suite
- JDBC-ODBC Bridge

JDBC
Treiber Typen



Quelle: https://de.wikipedia.org/wiki/Java_Database_Connectivity

JDBC
Connection Pool



JDBC Installation
Treiber für MySQL (Connector/J)

- Treiber und Dokumentation auf
  https://dev.mysql.com/doc/connector-j/en/connector-j-installing.html

- Empfohlene Installation durch Maven pom.xml

```
<dependencies>
    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.????</version>
    </dependency>
</dependencies>
```

JDBC, erstes Beispiel
Main Programm

```
public class JdbcTest {

    private static final Logger LOGGER = Logger.getLogger(JdbcTest.class.getName());
    private static final String CONNECTION_STRING =
        "jdbc:mysql://localhost/GMCD?user=gmaitre&password=Gilles";

    public static void main(String[] cmdArg) {
        try (Connection connection = DriverManager.getConnection(CONNECTION_STRING);
             Statement stm = connection.createStatement()) {
            displayResult(stm);
        } catch (SQLException ex) {
            LOGGER.log(Level.SEVERE, ex.getLocalizedMessage());
        }
    }

...
```

JDBC, erstes Beispiel
Abfrageresultate anzeigen

```
...

    private static void displayResult(Statement stm) throws SQLException {
        final String QUERY_STR = "SELECT * FROM CD";
        try (ResultSet rs = stm.executeQuery(QUERY_STR)) {
            while (rs.next()) {
                int cdid = rs.getInt("ID");
                String datum = rs.getString("Datum");
                String beschreibung = rs.getString("Beschreibung");
                System.out.println(MessageFormat.format(
                        "CDID: ''{0}'', Datum: ''{1}'', Beschreibung: ''{2}''",
                        cdid, datum, beschreibung));
            }
        }
    }
}
```

```
CDID: '1', Datum: '1968-01-13', Beschreibung: 'Yellow Submarine'
CDID: '2', Datum: '1975-01-24', Beschreibung: 'The Koeln Concert'
CDID: '3', Datum: '2009-09-04', Beschreibung: 'Dresden'
```

JDBC, erstes Beispiel
Welcher Port?

- `--port=port_num, -P port_num`

Option Sets Variable	Yes, **port**
Variable Name	**port**
Variable Scope	Global
Dynamic Variable	No
Value Set	**Type** numeric **Default** 3306

The port number to use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the **root** system user.

Siehe auch /etc/mysql/my.cnf

JDBC, erstes Beispiel
Installationsprobleme mit IntelliJ

1) Passen Sie auf, dass Ihr lokaler Firewall die Aktualisierung Ihrer IntelliJ-Umgebung nicht verhindert. Nach Bedarf TEMPORÄR das Firewall ausschalten.

2) Schreiben Sie IMMER in Ihrer Maven pom.xml Datei die Java JDK-Version, die Sie verwenden (als Property).
   Mehr Informationen hier.
   Überprüfen:
   a) Unter Menü Setting / Build... / Compiler / Java Compiler sollte jetzt der Wert von "Per module bytecode version" gleich dem pom.xml Wert sein. Z.B. 1.11 (und auf keinem Fall 1.5 sein. Dies ist ein Bug von IntelliJ).
   b) Unter Menü Project structure / Project sind die Werte von "Project SDK" sowie "Project Language Level", gleich dem pom.xml Wert.
   c) Unter Menü Project structure / Modules ist der Wert von "Language Level", gleich dem pom.xml Wert.

3) Falls Sie wegen der TimeZone vom MySQL-Server eine Runtime-Exception bekommen, am einfachsten im MySQL Connection-String folgendes anhängen: "&serverTimezone=UTC".
   Mehr Informationen hier.

JDBC, erstes Beispiel
Verbindungsprobleme

Häufige Fehler:

- Error 2003: Can't connect to MySQL server
  => Zugriff nicht möglich, siehe my.cnf
  (Problem mit Hostname, Port oder DBName?)

- Error 1130: Host 'xyz' is not allowed to connect to this MySQL server
  => Siehe mysql.user, Spalte "hostname"

- Error 1045: Access denied for user 'gugus@xyz'
  => Falscher Name oder Passwort ;-)

JDBC
Java, JDBC und MySQL Typen (1)

Java Data Type	MySQL Column Type
boolean, Boolean	BIT(1)
byte, Byte	BIT(1) to BIT(8), TINYINT
short, Short	BIT(1) to BIT(16), SMALLINT, YEAR
int, Integer	BIT(1) to BIT(32), INT
long, Long	BIT(1) to BIT(64), BIGINT, BIGINT UNSIGNED
float, Float	FLOAT
double, Double	DOUBLE
java.math.BigDecimal	NUMERIC, DECIMAL
java.math.BigInteger	NUMERIC (precision = 0), DECIMAL (precision = 0)

Quelle: https://dev.mysql.com/doc/ndbapi/en/mccj-using-clusterj-mappings.html

JDBC
Java, JDBC und MySQL Typen (2)

Java Data Type	MySQL Column Type
Java.util.Date	DATETIME, TIMESTAMP, TIME, DATE
Java.sql.Date	DATE
Java.sql.Time	TIME
Java.sql.Timestamp	DATETIME, TIMESTAMP

Java Data Type	MySQL Column Type
String	CHAR, VARCHAR, TEXT
byte[]	BINARY, VARBINARY, BLOB

Quelle: https://dev.mysql.com/doc/ndbapi/en/mccj-using-clusterj-mappings.html

JDBC Grundlagen
ExecuteQuery

```
/**
 * Executes the given SQL statement, which returns a single
 * ResultSet object.
 */
ResultSet executeQuery(String sql) throws SQLException;

// Beispiel
ResultSet rs = stm.executeQuery
                          ("SELECT PersNr, Name FROM Autor");
```

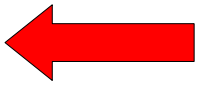
JDBC Grundlagen
ExecuteUpdate

```
/**
 * Executes the given SQL statement, which may be an INSERT,
 * UPDATE, or DELETE statement or an SQL statement that returns
 * nothing, such as an SQL DDL statement.
 * @return either
 *   (1) the row count for (DML) statements
 *   (2) 0 for SQL statements that return nothing
 */
int executeUpdate(String sql) throws SQLException;

// Beispiel
int nbrOfRows = stm.executeUpdate
       ("update Autor set Vorname='Gilles J.F.' " +
        "where Vorname='Gilles';");
```

JDBC Grundlagen
Werte aus einem ResultSet extrahieren

```
ResultSet rs = stm.executeQuery("SELECT PersNr, Name FROM Autor");
while (rs.next()) {
    int key = rs.getInt("PersNr");
    String name = rs.getString("Name");
}
rs.close();
```



```
ResultSet rs = stm.executeQuery("SELECT PersNr, Name FROM Autor");
while (rs.next()) {
    int key = rs.getInt(1);
    String name = rs.getString(2);
}
rs.close();
```

Anderer Weg:
- getString + explizite Konvertierung
- getObject + Casting

JDBC Grundlagen
AUTO_INCREMENT Werte finden

```
/**
 * Retrieves any auto-generated keys created as a result of
 * executing this Statement object.
 * If this Statement object did not generate any keys,
 * an empty ResultSet object is returned.
 */
ResultSet getGeneratedKeys() throws SQLException;
```

JDBC Grundlagen
Metadaten

- Wenn "ResultSet" nicht voraussehbar ist
- Klasse "ResultSetMetaData" studieren
- All what you ever wanted to know about a "ResultSet"...
- Es gibt auch eine Klasse *DatabaseMetaData*

JDBC Grundlagen
0, null und NULL

- "executeQuery" gibt ein "ResultSet" zurück
  - Kann leer sein
  - next() verwenden
- In einem "ResultSet" kann ein Element 0 oder NULL sein
  - 0 ist 0 ;-)
  - NULL => mit wasNull() testen

JDBC Grundlagen
Navigation im ResultSet

- next(), previous()
- first(), last(), isFirst(), isLast()
- beforeFirst(), afterLast()
- getRow()
- absolute(int row) // (1 <=> first; -1 <=> last)

JDBC Grundlagen
PreparedStatement (1)

- Wenn Abfrage mehrmals ausgeführt werden muss
- Wird (*eventuell*) vom Server "vor-kompiliert"
  => effizienter
- 1) Abfrageparameter mit "?" setzen
- 2) Parameter mit Werten setzen und Statement ausführen

JDBC Grundlagen
PreparedStatement (2)

```
PreparedStatement ps = con.prepareStatement
    ("SELECT * FROM Autor WHERE Vorname= ? ");

ps.setString(1, "Michael");

ResultSet rs = ps.executeQuery();
while (rs.next()) {
    String name = rs.getString("Name");
    System.out.println("Name: " + name);
}
rs.close();
```

JDBC Grundlagen
Transaktionen

- Reihenfolge von zusammengehörigen Operationen
- Wechsel zwischen konsistenten Zuständen
- Muss ACID-Eigenschaften erfüllen

```
con.setAutoCommit(false);

Statement stm = con.createStatement();
try {
    stm.executeUpdate
        ("INSERT Autor (PersNr, Name, Vorname) " +
         "VALUES (78, 'Wall', 'Larry')");

    stm.executeUpdate
        ("INSERT Autor (PersNr, Name, Vorname) " +
         "VALUES (90, 'Schneier', 'Bruce')");
    con.commit();

} catch (Exception e) {
    e.printStackTrace();
    con.rollback();  // Nach Savepoint 'X' wäre möglich
}
```

JDBC Grundlagen
Batches

Effizienter als einzelne Befehle, *im Prinzip*

```
Statement stm = con.createStatement();

stm.addBatch("UPDATE Autor SET PersNr='21' WHERE PersNr='12'");
stm.addBatch("UPDATE Autor SET PersNr='43' WHERE PersNr='34'");
stm.addBatch("UPDATE Autor SET PersNr='65' WHERE PersNr='56'");
stm.addBatch("UPDATE Autor SET PersNr='87' WHERE PersNr='78'");

int [] changes = stm.executeBatch();

for (int i = 0; i < changes.length; i++) {
    System.out.println
        ("Columns changed by stm: " + i + ": " + changes[i]);
}
```

JDBC Grundlagen
Batches *und* Transaktionen

```
con.setAutoCommit(false);

Statement stm = con.createStatement();

try {
    stm.addBatch("UPDATE Autor SET PersNr='12' WHERE PersNr='21'");
    stm.addBatch("UPDATE Autor SET PersNr='34' WHERE PersNr='43'");
    stm.addBatch("UPDATE Autor SET PersNr='56' WHERE PersNr='65'");
    stm.addBatch("UPDATE Autor SET PersNr='78' WHERE PersNr='87'");
    stm.executeBatch();
    con.commit();
} catch (Exception e) {
    e.printStackTrace();
    con.rollback();
}
```

JDBC Grundlagen
Mit BLOBs arbeiten

- In Selbststudium
- Ein gutes Beispiel ist hier zu finden

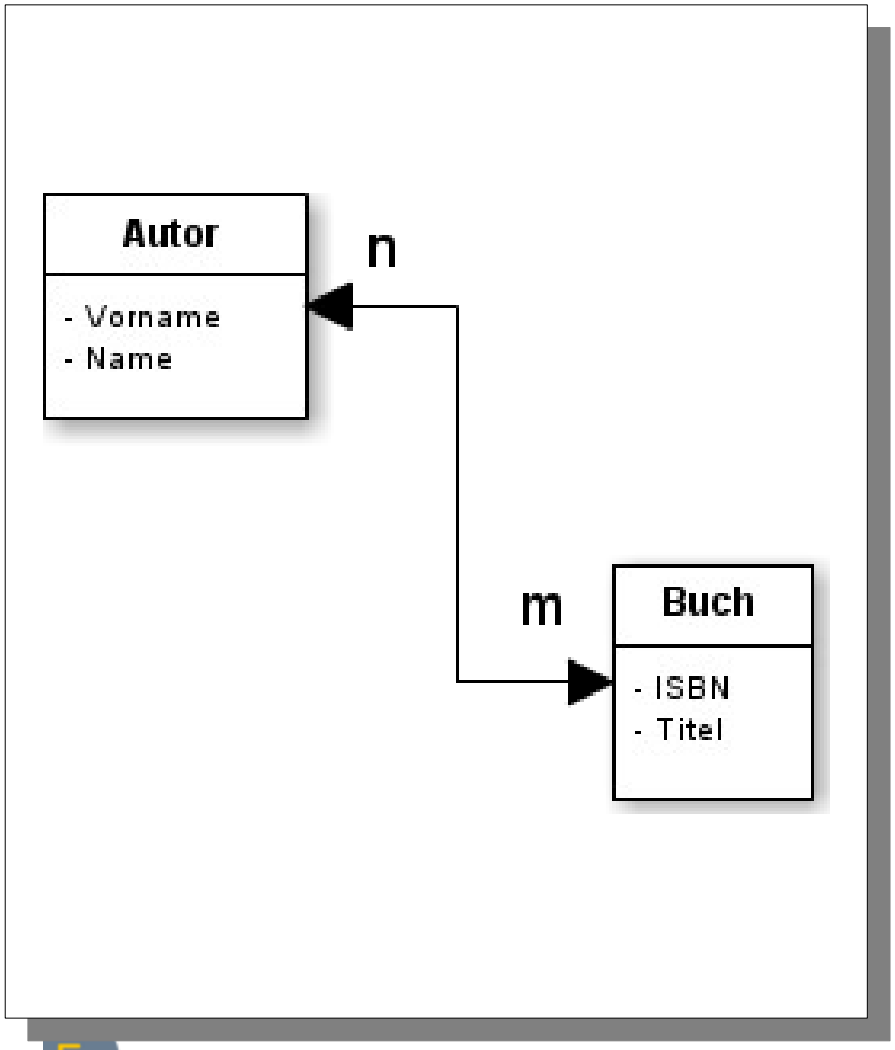
  http://www.java2s.com/Code/Java/Database-SQL-JDBC/InsertpicturetoMySQL.htm

JavaDB
Was ist das?

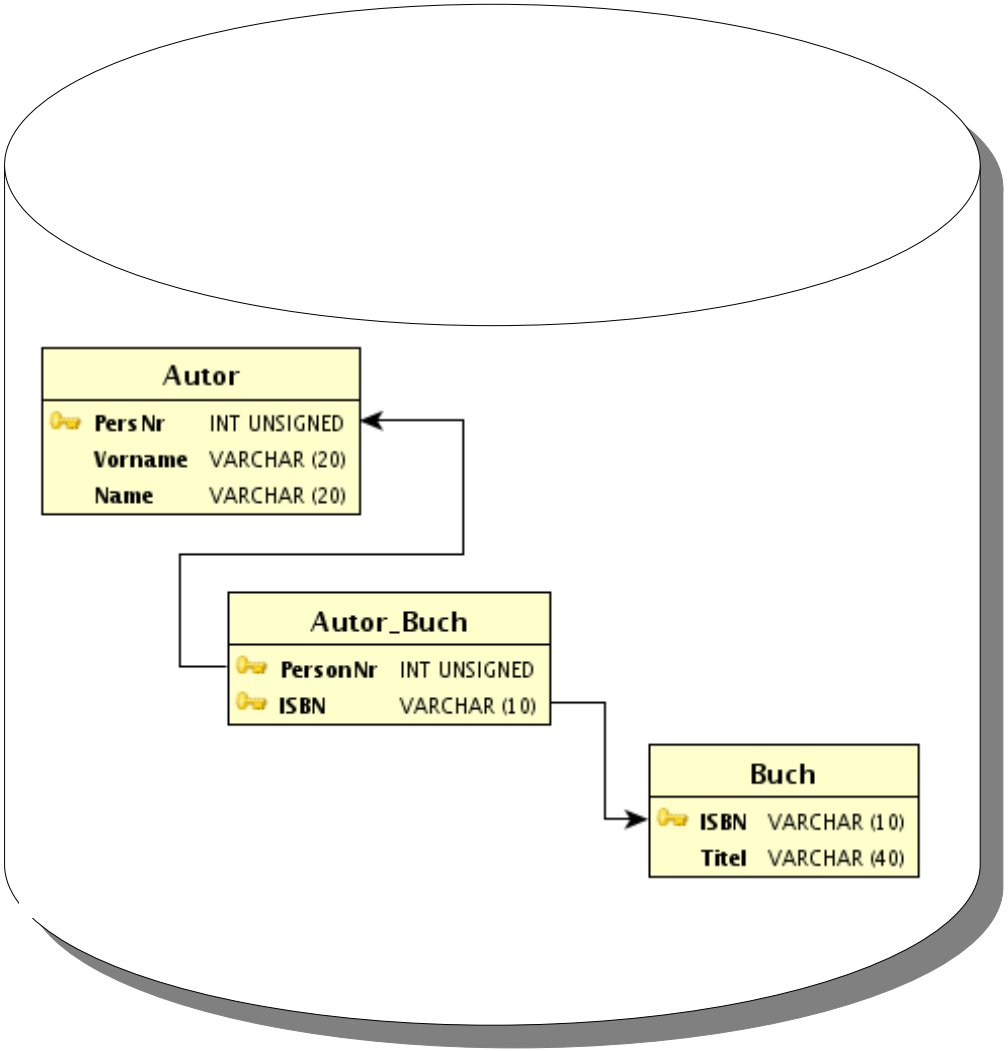
- Siehe http://www.oracle.com/technetwork/java/javadb/overview/index.html
- Praktisch für "kleine" Projekte, die eine SQL-DB brauchen
- JavaDB = Apache Derby im Java JDK integriert (separates Download)
- Tools:
  - java -jar $DERBY_HOME/lib/derbyrun.jar sysinfo
  - java -jar $DERBY_HOME/lib/derbyrun.jar ij
  - java -jar $DERBY_HOME/lib/derbyrun.jar dblook -d 'jdbc:derby:firstdb'

JPA / ORM
Was ist das?

Applikation



Relationale Datenbank



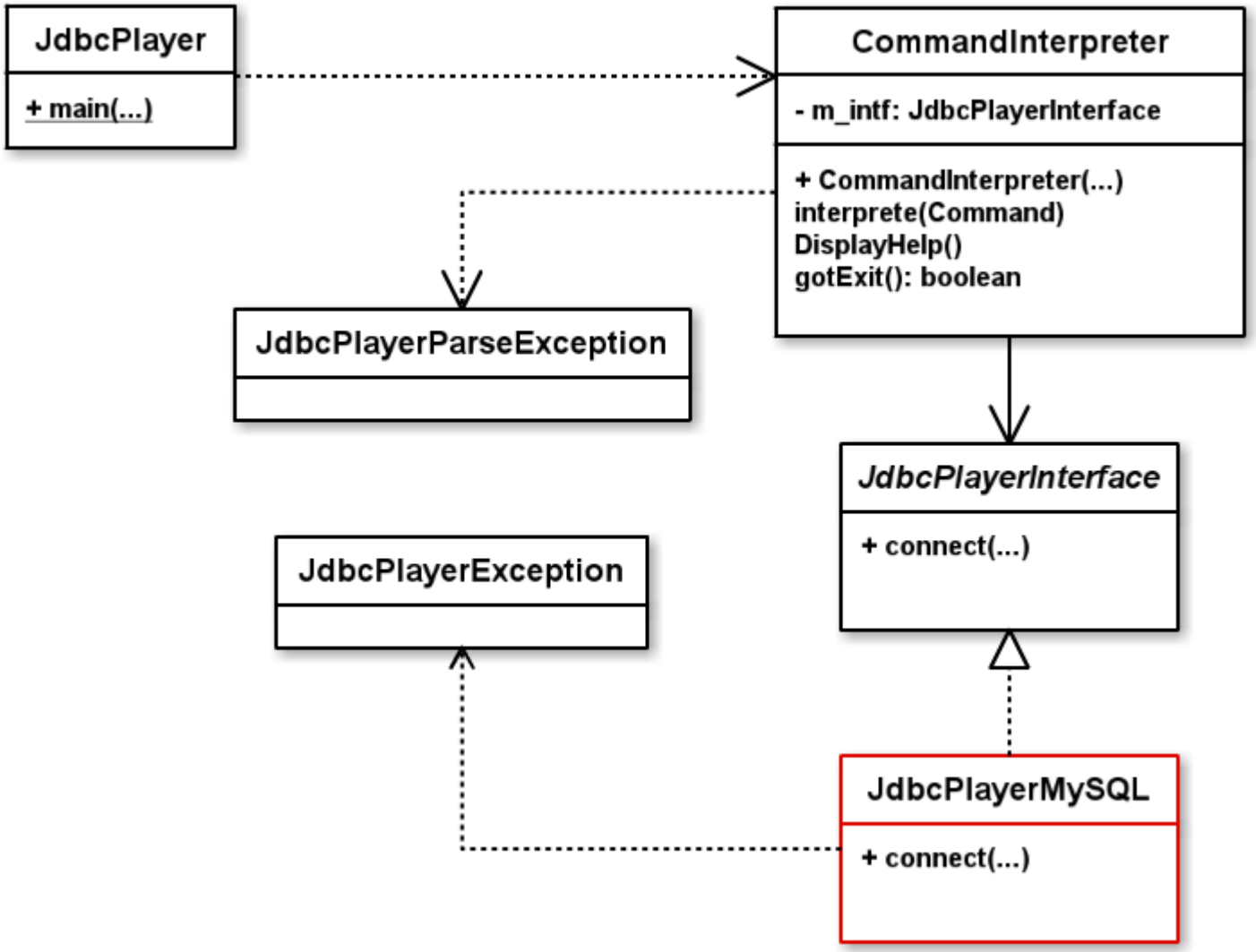
Datenbank-Migration
Flyway

- Ziel: Konsistenz zwischen Code und DB-Schema sicherstellen
- Weg dazu: DB-Schema versionieren und mit dem Code verknüpfen
- Das Tool Flyway hilft
  - Why database migrations?
    Siehe https://flywaydb.org/getstarted/why
  - How Flyway works?
    Siehe https://flywaydb.org/getstarted/how

Übungen

1) Bringen Sie mein JdbcTest-Programm zum Laufen.
   a) Den Code finden Sie unter https://web.mtg1.bfh.science/SD-RDB/JdbcTest.zip.
   b) Importieren Sie das Projekt in IntelliJ.
2) Erweitern Sie die Klasse "JdbcPlayerMySQL", die die "JdbcPlayerInterface" für GMCD-Datenbank realisiert (siehe nächste Seite).
   a) Den Code finden Sie unter https://web.mtg1.bfh.science/SD-RDB/JdbcPlayer.zip.
   b) Importieren Sie das Projekt in IntelliJ.
   c) Erweitern Sie die Klasse "JcbcPlayerMySQL.java.
   d) Fügen Sie neue Daten in die GMCD-Datenbank mit JdbcPlayer hinzu.

Das Programm JdbcPlayer
UML Klassendiagramm



Das Programm JdbcPlayer
Anwendungsbeispiel

```
JdbcPlayer Command Interpreter V1.2
-----------------------------------

Type 'help' to list the commands
JdbcPlayer> help

List of available commands (case-insensitive)
--------------------------------------------
connect "<jdbcConnectionString>"
disconnect
show Tables
execute query "<QueryString>"
execute update "<UpdateString>"
exit
help

JdbcPlayer> connect "jdbc:mysql://localhost/GMCD?user=...&password=..."
INFO: New connection eshtablished
JdbcPlayer> execute query "select * from Autor"

+----+------------+-------------------+
| ID | Datum      | Beschreibung      |
+----+------------+-------------------+
|  1 | 1968-01-13 | Yellow Submarine  |
|  2 | 1975-01-24 | The Koeln Concert |
|  3 | 2009-09-04 | Dresden           |
+----+------------+-------------------+

JdbcPlayer> exit
```