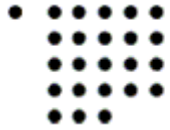


JavaDatabaseConnectivity

Übersicht

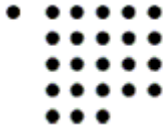
- Was ist JDBC?
- Geschichte und Ziele der JDBC API
- Connection, Statement und ResultSet
- Arrays, BLOBs, Objects
- Exceptions, Transaktionen und Meta Daten



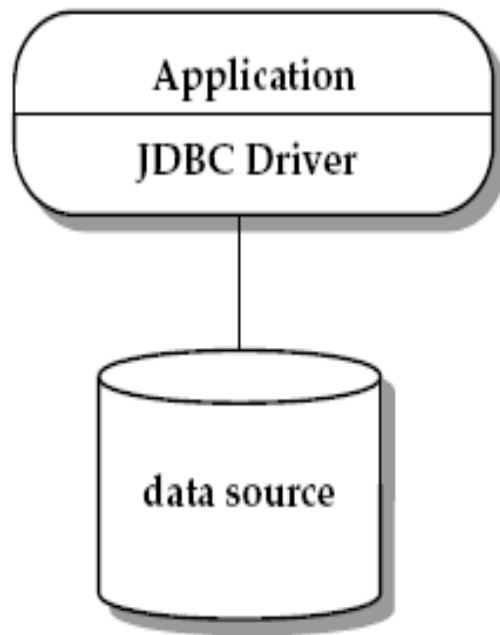
JavaDataBaseConnectivity

Was ist JDBC?

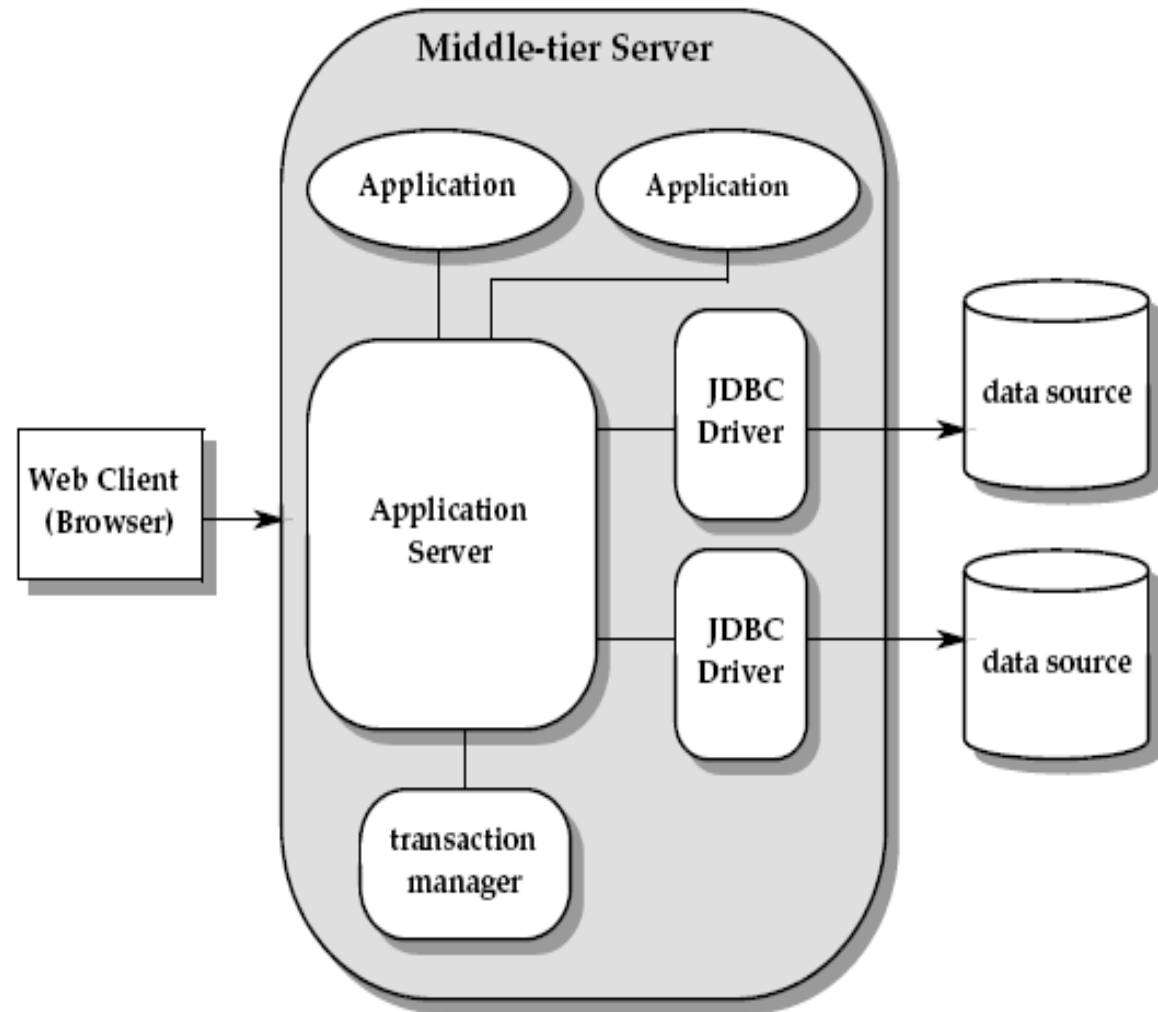
- Entwicklung von Sun microsystems
- JDBC API im Januar 1997 spezifiziert
- Ein Basic-Call-Interface für SQL-Datenbanken
- Besteht aus Klassen und Schnittstellen
- Plattformunabhängig
- Aktuell in der Version 4.0



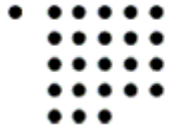
JavaDataBaseConnectivity



Two-tier Model



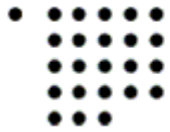
Three-tier Model



Java DataBase Connectivity

Geschichte

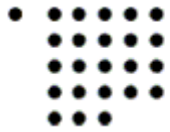
- JDBC 1 definiert Basisfunktionalität von JDBC
 - Herstellen von Verbindungen zu SQL-Datenbanken
 - Ausführen von SQL-Anweisungen
 - Verarbeiten von Anfrageergebnissen
- JDBC 2 erweitert JDBC 1 um
 - scroll- und änderbare Ergebnismengen
 - Batch-Änderungen
 - SQL:1999-Datentypen
 - benutzerdefinierte Typabbildungen
- JDBC 3 unterstützt weitere SQL:1999-Funktionalität
 - Sicherungspunkte
 - Änderungsmethoden für Spalten von konstruierten bzw. benutzerdefinierten Datentypen



Java Database Connectivity

Wichtigsten Änderungen in 4.0

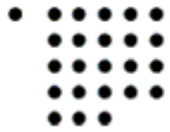
- Automatisches Laden von `java.sql.Driver`
- Vereinfachung des Development API
- ROWID data type hinzugekommen
- National Character Set Conversion Support
- Enhanced Support für BLOBs und CLOBs
- SQL/XML und XML Support
- Neue Funktionen wie `createBlob()` hinzugekommen



Java Database Connectivity

Ziele der JDBC API 4.0

- Fit into the Java EE and Java SE platforms
- Be consistent with SQL:2003
- Maintain the focus on SQL
- Keep it simple
- Enhance reliability, availability, and scalability



JavaDataBaseConnectivity

1. Verbindungsaufbau

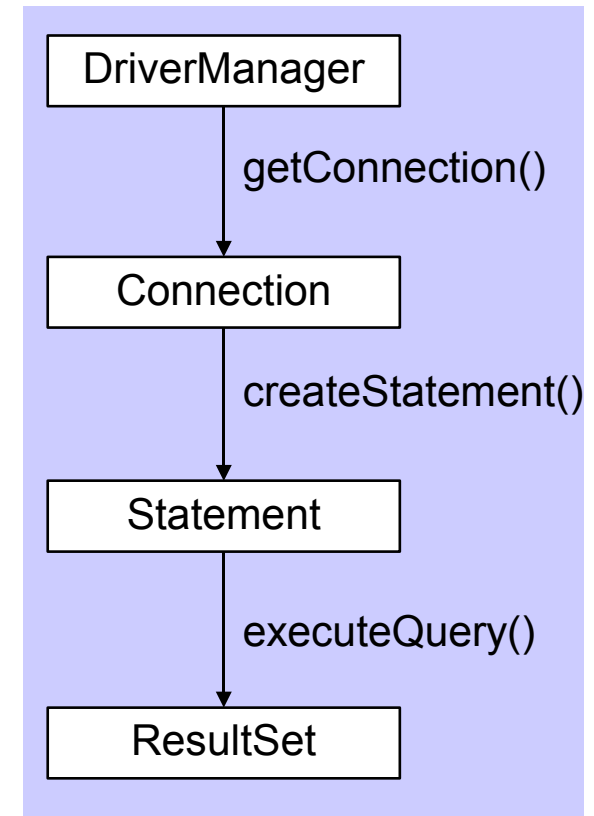
- Angabe der Verbindungsinformationen (und Treiber laden)

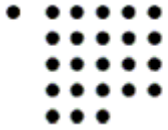
2. Senden einer SQL-Anweisung

- Definition der Anweisung
- Belegung von Parametern

3. Verarbeitung der Ergebnisse

- Navigation über Ergebnisrelation
- Zugriff auf Spalten
- Verarbeitung der Metadaten





JavaDataBaseConnectivity

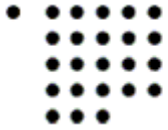
```
import java.sql.*;
import java.util.*;

public class Studenten {
    public static void main (String args []) {

        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        String url = "jdbc:<Verbindungsprotokoll>:<Port>:<Datenbankname>";
        Connection con = DriverManager.getConnection(url, "user", "pass");

        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT Mnr, Name FROM Studenten");

        while (rs.next()) {
            System.out.println("Matrikelnr : " + rs.getInt(1) + ", Name : " + rs.getString(2));
        }
        rs.close(); stmt.close(); con.close();
    }
}
```



JavaDataBaseConnectivity

Treiber laden (2 Varianten)

```
Class.forName("com.company.DBDriver");
```

```
DriverManager.registerDriver(new com.company.DBDriver());
```

Oracle

oracle.jdbc.driver.OracleDriver

DB2

COM.ibm.db2.jdbc.app.DB2Driver

Informix

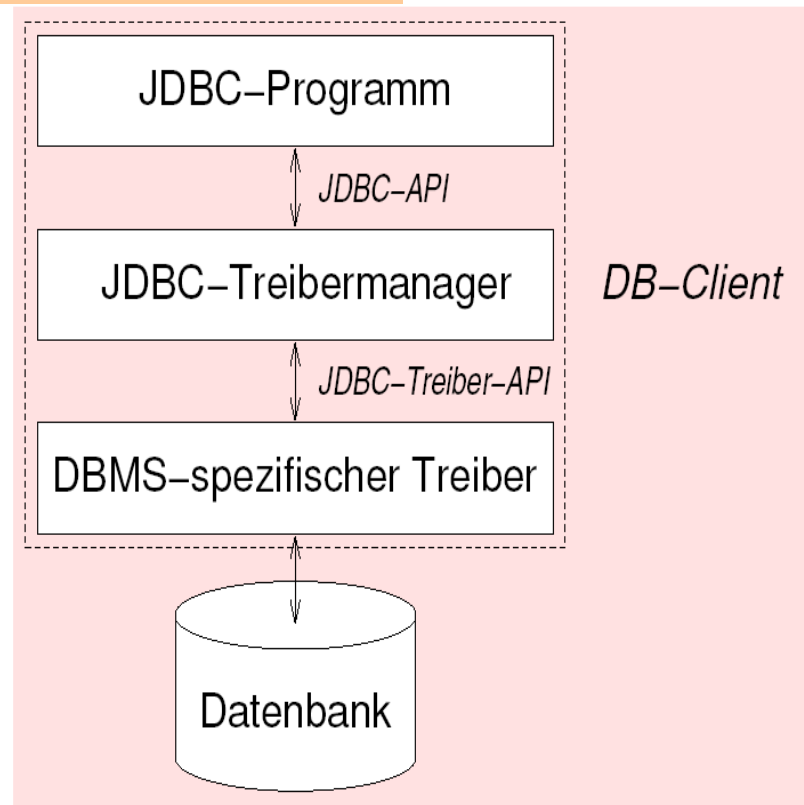
com.informix.jdbc.IfxDriver

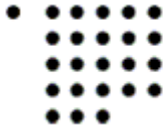
PostgreSQL

org.postgresql.Driver

MySQL

org.gjt.mm.mysql.Driver



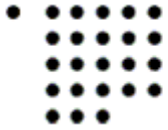


JavaDataBaseConnectivity

Verbindung herstellen

```
String url = "jdbc:oracle:thin:@oras1.gm.fh-koeln.de:1521:oras1";  
String user = "ORDBS_X";  
String pass = "ORDBS_X";  
Connection con = DriverManager.getConnection(url, user, passwd);
```

- JDBC-URL spezifiziert
 - Datenquelle/Datenbank
 - Verbindungsmechanismus (Protokoll, Server-Host und Port)



Java DataBase Connectivity

Anweisungsobjekt (Statement) erzeugen

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

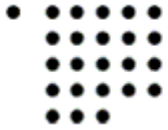
Anweisung ausführen

```
String queryStmt = "SELECT Name, Anschrift FROM Studenten";  
ResultSet rset = stmt.executeQuery(queryStmt);
```

– Klasse `java.sql.Statement`

- Ausführung von Anfragen (**SELECT**) mit `executeQuery`
- Ausführung von Änderungen (**DELETE**, **INSERT**, **UPDATE**) mit `executeUpdate` (liefert Anzahl der betroffenen Tupel/Objekte)

```
String delStmt = "DELETE Studenten WHERE MNr = 110xxxxx";  
int rows = stmt.executeUpdate(delStmt);
```



JavaDatabaseConnectivity

Navigation über Ergebnismenge (Cursor-Prinzip)

```
while (rset.next()) {  
    // Verarbeitung der einzelnen Tupel  
    ...  
}
```

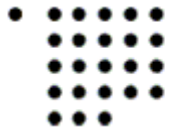
Zugriff auf Spaltenwerte über ~~getXXX~~-Methoden

- über Spaltenindex

```
String name = rset.getString(1);
```

- über Spaltenname

```
String name = rset.getString("Name");
```



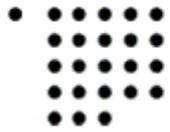
JavaDataBaseConnectivity

Typabbildung von SQL nach Java

Typabbildung von Java nach SQL

SQL	get-Methode	Java-Typ
SMALLINT	getShort	short
INTEGER	getInt	int
BIGINT	getLong	long
NUMERIC	getBigDecimal	BigDecimal
DECIMAL	getBigDecimal	BigDecimal
FLOAT	getDouble	double
REAL	getFloat	float
DOUBLE PRECISION	getDouble	double
CHAR	getString	String
VARCHAR	getString	String
BIT	getBoolean	boolean
VARYING BIT	getBytes	byte[]
DATE	getDate	Date
TIME	getTime	Time
TIMESTAMP	getTimestamp	Timestamp

Java-Typ	set-Methode	SQL
byte	setByte	SMALLINT
short	setShort	SMALLINT
int	setInt	INTEGER
long	setLong	BIGINT
BigDecimal	setBigDecimal	NUMERIC
float	setFloat	REAL
double	setDouble	DOUBLE
String	setString	VARCHAR
boolean	setBoolean	BIT
byte[]	setBytes	VARYING BIT
Date	setDate	DATE
Time	setTime	TIME
Timestamp	setTimestamp	TIMESTAMP



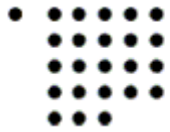
JavaDataBaseConnectivity

java.sql.PreparedStatement

```
String insStmt = "INSERT INTO Kunde VALUES (?, ?)";  
PreparedStatement stmt = con.prepareStatement(insStmt);  
  
stmt.setString(1, "George");  
stmt.setFloat(2, 4500.00);  
  
stmt.executeUpdate(insStmt);
```

java.sql.CallableStatement

```
String callStmt = "{CALL TestProc(?, ?)}";  
CallableStatement stmt = con.prepareCall(callStmt);  
stmt.setInt(1, 42);  
  
stmt.registerOutParameter(2, java.sql.Types.FLOAT);  
stmt.executeUpdate();  
double res = stmt.getDouble(2);
```



JavaDataBaseConnectivity

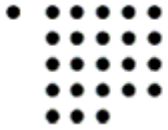
Typabbildung von SQL nach Java

Typabbildung von Java nach SQL

SQL	get-Methode	Java-Typ
BLOB	getBlob	Blob
CLOB	getClob	Clob
ARRAY	getArray	Array
ROW	getObject	Struct
REF	getRef	Ref
Distinct-Typ	getX	X
Strukturierter Typ	getObject	Struct

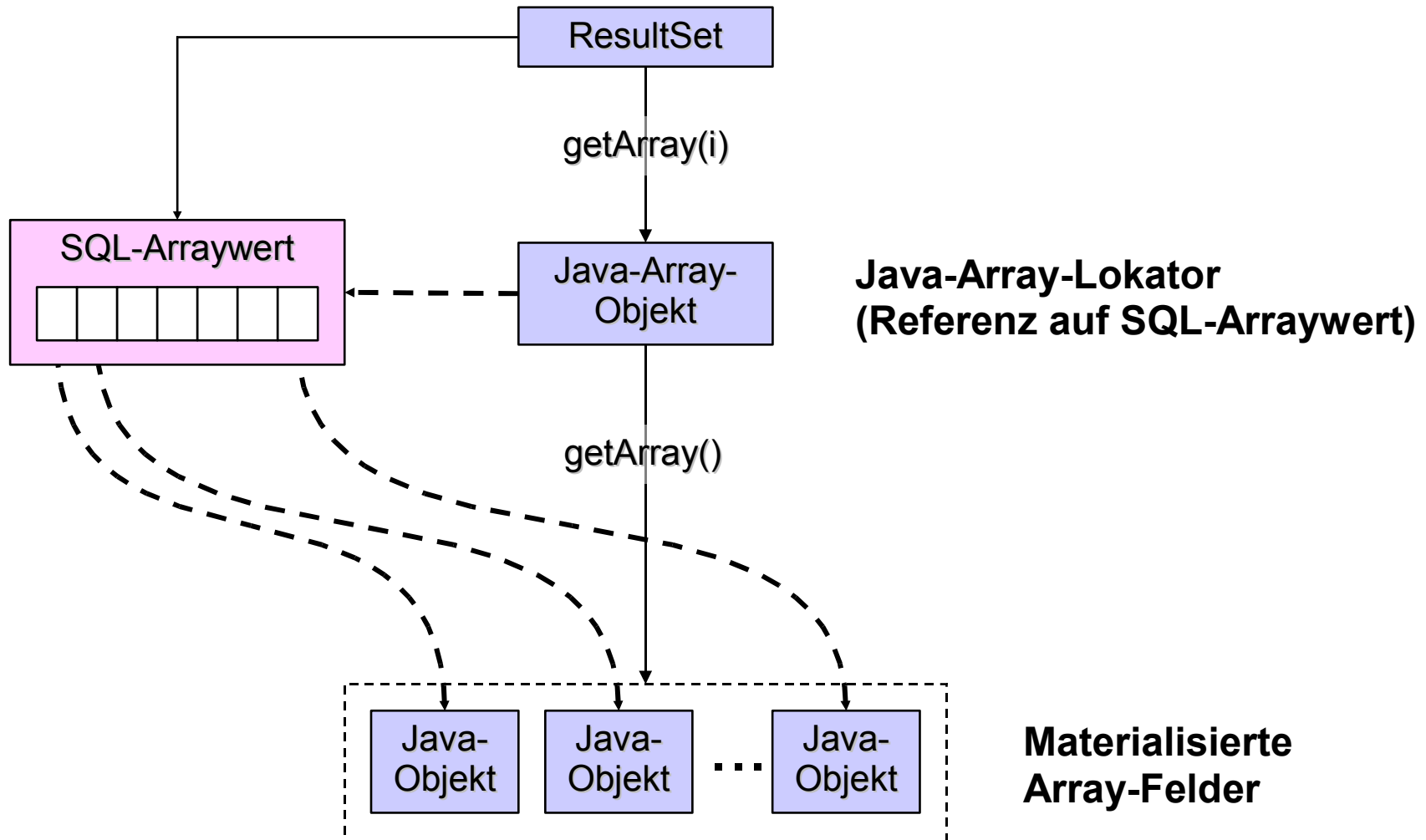
Java-Typ	set-Methode	SQL
Blob	setBlob	BLOB
Clob	setClob	CLOB
Array	setArray	ARRAY
Struct	setObject	ROW
Ref	setRef	REF
X	setX	Distinct-Typ
Struct	setObject	Strukturierter Typ

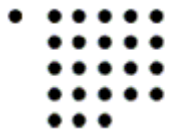
X ist der Basisdatentyp, der einem Distinct-Typ zugrunde liegt!



JavaDataBaseConnectivity

Illustration der Abbildung von SQL-Arrays nach Java





JavaDataBaseConnectivity

Verarbeiten von SQL-Array (1)

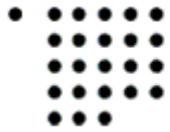
```
String str = "SELECT Name, Telefone FROM Studenten";
ResultSet rs = stmt.executeQuery(str);

while (rs.next())
{
    // lies SQL-Array in Java-Array-Objekt
    Array tarray = rs.getArray(2);

    // extrahiere Arrayfelder
    String[] telNr = (String[]) tarray.getArray();

    // gib Namen des Studenten aus
    System.out.println("Telefonnummern von " + rs.getString(1));

    // gib Arrayfelder aus
    for(int i = 0; i < telNr.length; i++)
        System.out.println(i + ". Nummer: " + telNr[i]);
}
```



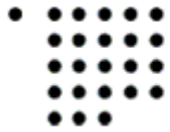
JavaDataBaseConnectivity

Verarbeiten von SQL-Array (2)

```
// erzeuge Java-Repräsentant für einen SQLArraywert
Object [] telefonnummern = new Object [2];
telefonnummern[0] = "004116321172";
telefonnummern[1] = "00493916712020";
Array telArray = new Array(telefonnummern);

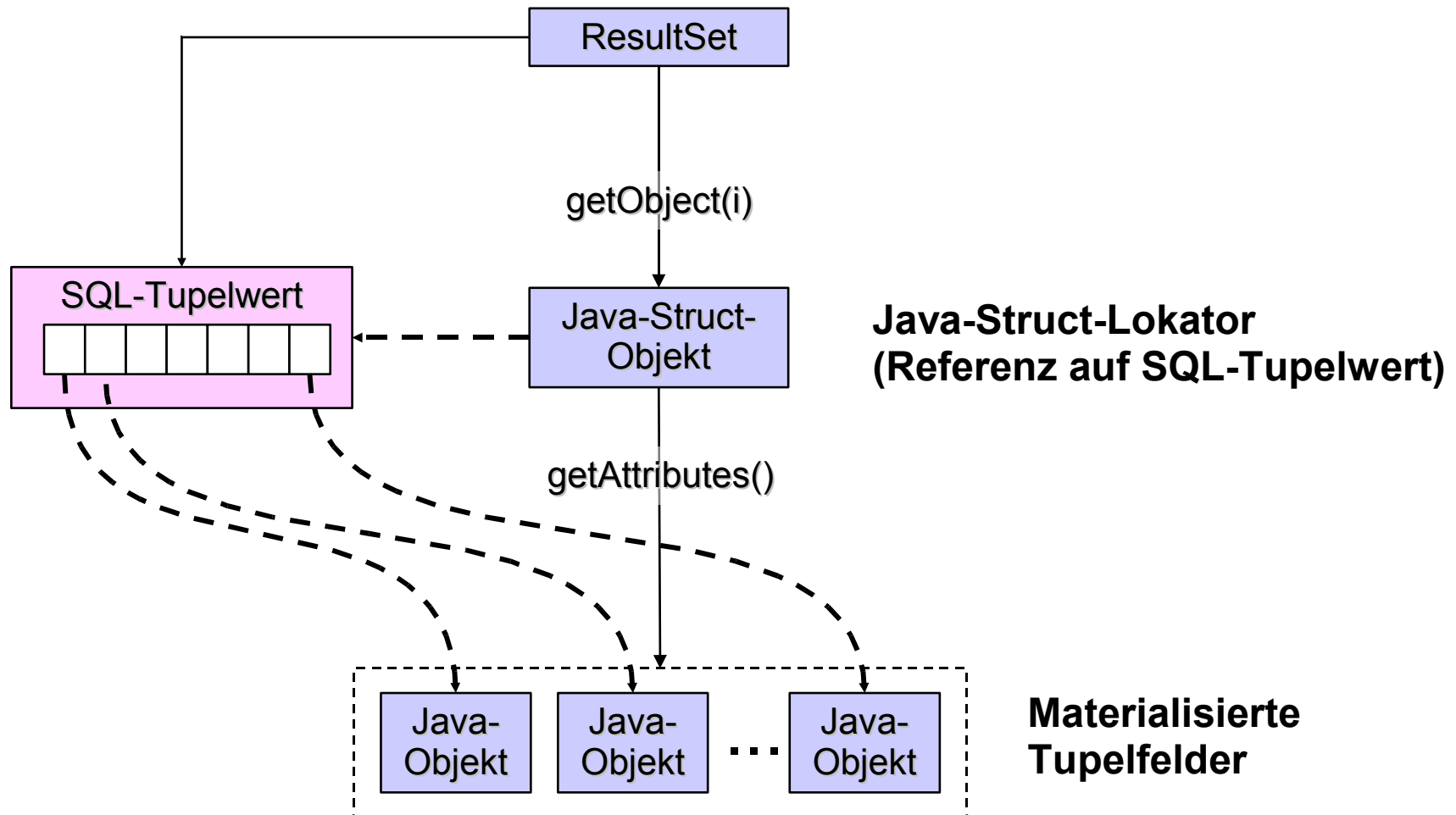
// erzeuge vorbereitetes Anweisungsobjekt
String str = "UPDATE Studenten SET Telefone = ? WHERE Mnr = 110xxxxx";
PreparedStatement ps = con.prepareStatement(str);

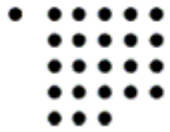
// belege arraywertige Spalte und führe Änderung aus
ps.setArray(1, telArray);
ps.execute();
```



JavaDataBaseConnectivity

Illustration der Abbildung von SQL-Objects nach Java





JavaDataBaseConnectivity

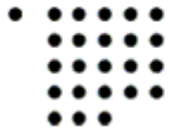
Verarbeiten von SQL-Objects (1)

```
String str = "SELECT Name, Anschrift FROM StudentenTupeltabelle";
ResultSet rs = stmt.executeQuery(str);

while (rs.next())
{
    // lies tupelwertige Spalte in ein Java-Struct-Objekt ein
    Struct astruct = (Struct) rs.getObject(2);

    // extrahiere Tupelfelder
    Object atts [] = astruct.getAttributes();

    // gib Kundenname und Wohnort aus
    System.out.println(rs.getString(1) + " wohnt in " + atts[3]);
}
```



JavaDataBaseConnectivity

Verarbeiten von SQL-Objects (2)

```
// erzeuge Java-Repräsentant für einen SQL-Tupelwert
```

```
Object adresse [] = new Object [5];  
adresse[0] = "Über der Aar";      // Strasse  
adresse[1] = 19;                  // Nr  
adresse[2] = 65307;              // PLZ  
adresse[3] = "Bad Schwalbach";   // Ort  
adresse[4] = "D";                // Land
```

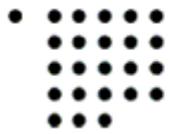
```
Struct adresseStruct = new Struct(adresse);
```

```
// erzeuge vorbereitetes Anweisungsobjekt
```

```
String str = "UPDATE StudentenTupeltabelle " + "SET Anschrift = ? WHERE MNr = xxx";  
PreparedStatement ps = con.prepareStatement(str);
```

```
// belege tupelwertige Spalte und führe Änderung aus
```

```
ps.setObject(1, adresseStruct);  
ps.execute();
```



JavaDataBaseConnectivity

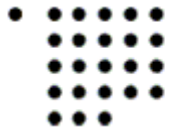
Verarbeiten von SQL-BLOBs

```
String str = "SELECT Mnr, Bild FROM Student UNNEST(Bilder) b (Bild)";
ResultSet rs = stmt.executeQuery(str);

while(rs.next()){
    // lies SQL-BLOB in Java-Blob-Object (erzeuge Blob-Locator)
    Blob blob = rs.getBlob(2);

    // erzeuge Puffer, um Blob-Daten partiell abzuarbeiten
    FileOutputStream out = new FileOutputStream("student1.jpg");
    BufferedInputStream in = new BufferedInputStream(blob.getBinaryStream());

    // kopiere byteweise von input nach output
    int b;
    byte[] puffer = new byte[1024];
    while ((b = in.read(puffer, 0 , 1024)) != -1){
        out.write(puffer, 0 , b);
    }
}
```



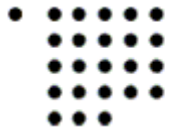
JavaDataBaseConnectivity

Fehlerbehandlung

```
try {  
    // Aufruf von JDBC-Methoden  
    ...  
} catch (SQLException exc) {  
    System.out.println("SQLException: " + exc.getMessage());  
}
```

Transaktionssteuerung

```
// AutoCommit-Modus ausschalten  
con.setAutoCommit(false);  
// DML-Anweisungen  
...  
// COMMIT ausführen  
con.commit();  
// Änderungen zurücknehmen  
con.rollback();
```



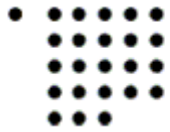
JavaDataBaseConnectivity

Meta Daten

```
ResultSetMetaData rsmd = rs.getMetaData();
int numberOfColumns = rsmd.getColumnCount();

for (int i=1;i<= numberOfColumns; i++) {
    String colName = rsmd.getColumnName(i);
    String tableName = rsmd.getTableName(i);
    String name = rsmd.getColumnTypeName(i);
    boolean caseSen = rsmd.isCaseSensitive(i);
    boolean writable = rsmd.isWritable(i);

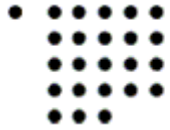
    System.out.println("" +tableName);
    System.out.println("" +colName);
    System.out.println("" +name);
    System.out.println("" +caseSen);
    System.out.println("" +writeable);
}
```



JavaDataBaseConnectivity

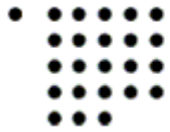
Quellen und Literatur

- [JDBC API 4.0 Specifications](#)
- dpunkt.verlag, Objektrelationale Datenbanken – Türker, Saake (Auflage 2006)
- <http://java.sun.com/products/jdbc/>



JavaDataBaseConnectivity

**Vielen Dank
für Ihre Aufmerksamkeit**



JavaDataBaseConnectivity

Aufgaben

- Aufgabe a)
 - Schreiben Sie eine Java-Klasse „Student“, die eine Connection zur Datenbank aufbaut (Treiber können Sie hier herunterladen).
- Aufgabe b)
 - Ergänzen Sie diese Klasse um das Holen von Daten eines Studenten aus der Datenbank (inkl. Telefonnummern-Array). Lassen Sie die Daten in der Konsole ausgeben. Verwenden Sie hierbei die verschiedenen Statement-Klassen.
- Aufgabe c)
 - Ergänzen Sie diese Klasse um ein einfaches Exception-Handling.