

JDBC

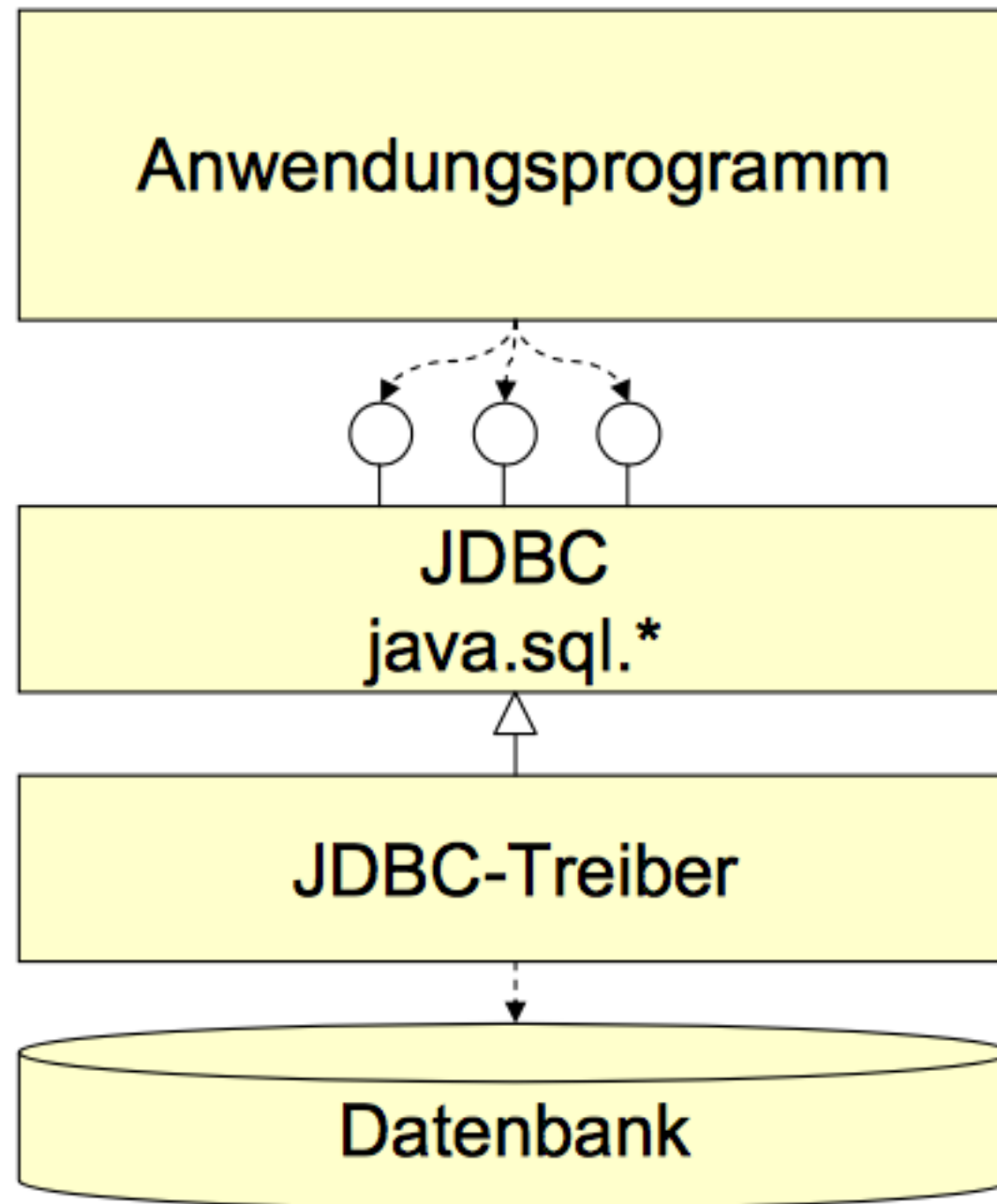
((itanius informatik))

Java Foundation Track by Carsten Bokeloh

JDBC

- Die Schnittstellen von
java.sql
- Treiber und
Verbindungsaufbau
- Statement
- ResultSet

JDBC



Die Schnittstelle java.sql

- ❖ enthält java Schnittstelle zu Datenbanken
- ❖ standardisierter Zugriff auf eine Datenbank, egal ob Oracle, Postgres, Mysql
- ❖ JDBC Treiber implementiert die JDBC Interfaces und stellt Zugriff auf konkretes DBMS
- ❖ sql-statements sollten natürlich auch standard-sql sein.

Was für Interfaces?

- ❖ Connection
 - ❖ Ermöglicht den Verbindungsaufbau zu einer Datenbank.
- ❖ Driver
 - ❖ Ein Interface, was jede Treiberklasse implementieren muss.
- ❖ PreparedStatement
 - ❖ ein Objekt, welches ein vorcompiliertes Statement repräsentiert
- ❖ ResultSet
 - ❖ ErgebnisCursor
- ❖ Statement
 - ❖ Objekt um ein Statement auszuführen

DriverManager

- ❖ Zentrale zur Verwaltung von Datenbanktreiber
- ❖ hat ausschließlich statische Methoden.
- ❖ möglich, Liste aller geladenen Treiber ausgeben zu lassen
- ❖ auch der Verbindungsaufbau **kann** über die Klasse DriverManager erfolgen.

Liste der verfügbaren Treiber

```
public static void printAllDrivers()
{
    int j=0 ;
    for( Enumeration en = DriverManager.getDrivers() ; en.hasMoreElements() ; j++)
        System.out.println( en.nextElement().getClass().getName() );

    if (j==0)
        System.out.println("Treiberliste leer");

    System.out.println("-----");
}
```


Treiber laden

```
try {  
//   Eine Möglichkeit zum Laden der Treiber ist die Verwendung  
//   der statische Methode Class.forName().  
//   Wird der Treiber nicht gefunden, so wirft die Methode eine  
    Class.forName("org.postgresql.Driver");  
  
} catch (ClassNotFoundException e) {  
  
    System.out.println("Where is your PostgreSQL JDBC Driver? "  
        + "Include in your library path!");  
    e.printStackTrace();  
    return;  
  
}  
  
System.out.println("PostgreSQL JDBC Driver Registered!");
```


Alternative fürs Treiber laden

```
// Eine Möglichkeit zum Laden der Treiber ist die Verwendung
// der statische Methode Class.forName().
// Wird der Treiber nicht gefunden, so wirft die Methode eine
//Treiber laden für Postgres
Class drv = org.postgresql.Driver.class;
```

Verbindungsaufbau über DriverManager

```
private static final String DB_CONNECTION = "jdbc:postgresql://127.0.0.1:5432/autovermietung";  
private static final String DB_USER = "tom";  
private static final String DB_PASSWORD = "password";
```

```
try {  
    dbConnection = DriverManager.getConnection(  
        DB_CONNECTION, DB_USER, DB_PASSWORD);  
    return dbConnection;  
} catch (SQLException e) {  
    System.out.println(e.getMessage());  
}
```


Verbindungsaufbau über ein DataSource-Object

- ❖ Sind Treiber geladen, so werden sie von der Klasse DriverManager registriert
- ❖ Ab Java 1.4 gibt es eine weitere Möglichkeit eine Verbindung zu einer Datenbank aufzubauen. Im Package javax.sql befindet sich Das Interface DataSource.

Verbindungsaufbau über DataSource

```
private static final String DB_USER = "tom";  
private static final String DB_PASSWORD = "password";  
private static final String SERVER = "localhost";  
private static final Integer PORT = 5432;  
private static final String DBNAME = "autovermietung";
```

```
try {  
    //besserer Weg eine Connection aufzubauen als über DriverManager.  
    PGSimpleDataSource dataSource = new PGSimpleDataSource();  
    if( dataSource instanceof DataSource)  
        System.out.println("PGSimpleDataSource implements the javax.sql.DataSource interface");  
  
    System.out.println("datasource created");  
    dataSource.setServerName(SERVER);  
    dataSource.setPortNumber(PORT);  
    dataSource.setDatabaseName(DBNAME);  
    dataSource.setUser(DB_USER);  
    dataSource.setPassword(DB_PASSWORD);  
  
    dbConnection = dataSource.getConnection();  
} catch (SQLException e) {  
    System.out.println(e.getMessage());  
}  
  
return dbConnection;
```

Übung

❖ UebJDBC / Uebung1

Statement

- ❖ Ein Java-Statement dient dazu, SQL-Statements auf die Datenbank abzusetzen. Dabei werden java-seitig nur zwei Arten von SQL-Statements unterschieden. Entweder handelt es sich um ein select-Statement oder um kein Select-Statement.

executeQuery

- ❖ Ein SELECT-Statement liefert einen Auszug aus einer Tabelle (oder aus mehreren Tabellen oder auch eine ganze Tabelle). Ein SELECT-Statement wird für gewöhnlich mit der Methode `public ResultSet executeQuery(String sql)` abgeschickt und liefert den Tabellenauszug in Form eines Objekts vom Typ `ResultSet` zurück.

executeUpdate

- ❖ Alle nicht SELECT-Statements kann man mit der Methode `public int executeUpdate(String sql)` abschicken. Mit `executeUpdate()` werden sowohl CREATE-Statements als auch Statements mit INSERT, DELETE, ALTER, RENAME oder UPDATE usw. angeschickt. Der Returnwert von `executeUpdate()` ist bei INSERT, UPDATE oder DELETE die Anzahl der betroffenen Zeilen, bei CREATE oder DROP ist er 0.
- ❖ Alle anderen `execute()` und `executeXXX()` Methoden sind für komplexe SQL-Statements zuständig.

executeQuery

```
String selectTableSQL = "SELECT * from auto";
Statement statement = connection.createStatement();
// Ein SELECT-Statement liefert einen Auszug aus einer Tabelle
// (oder aus mehreren Tabellen oder auch eine ganze Tabelle).
// Ein SELECT-Statement wird für gewöhnlich mit der Methode
// public ResultSet executeQuery(String sql) abgeschickt
// und liefert den Tabellenauszug in Form eines Objekts vom Typ ResultSet zurück.
// Das Interface ResultSet vereinbart mittlerweile mehr als 150 (!) Methoden
// um auf ein geliefertes ResultSet lesend oder schreibend zuzugreifen.
ResultSet rs = statement.executeQuery(selectTableSQL);
```

executeUpdate

```
Statement statement = connection.createStatement();  
// Alle nicht SELECT-Statements kann man mit der Methode  
// public int executeUpdate(String sql) abschicken.  
// Mit executeUpdate() werden sowohl CREATE-Statements als auch Statements mit INSERT,  
// DELETE, ALTER, RENAME oder UPDATE usw. geschickt.  
// Der Returnwert von executeUpdate() ist bei INSERT,  
// UPDATE oder DELETE die Anzahl der betroffenen Zeilen,  
// bei CREATE oder DROP ist er 0.  
statement.executeUpdate(insertTableSQL);
```


PreparedStatement

- ❖ Vorübersetzung für SQL-Anweisungen
- ❖ prepared deshalb, weil die Anweisungen in einem ersten Schritt zur Datenbank geschickt und dort in ein internes Format umgesetzt werden.
- ❖ Datenbank kann sie schneller ausführen, da sie in einem optimalen Format vorliegen
- ❖ Ein Geschwindigkeitsvorteil macht sich immer dann besonders bemerkbar, wenn Schleifen Änderungen an Tabellenspalten vornehmen.

PreparedStatement

```
// Ein Prepared Statement ist eine sogenannte vorbereitete Anweisung für ein Datenbanksystem.
// Im Gegensatz zu gewöhnlichen Statements enthält es noch keine Parameterwerte.
// Stattdessen werden dem Datenbanksystem Platzhalter übergeben.
//
// Mittels Prepared Statements können SQL-Injections effektiv verhindert werden,
// da das Datenbanksystem die Gültigkeit von Parametern prüft,
// bevor diese verarbeitet werden.
//
// Soll ein Statement mit unterschiedlichen Parametern mehrere Male
// (z. B. innerhalb einer Schleife) auf dem Datenbanksystem ausgeführt werden,
// können Prepared Statements einen Geschwindigkeitsvorteil bringen,
// da das Statement schon vorübersetzt im Datenbanksystem vorliegt und nur noch mit
// den neuen Parametern ausgeführt werden muss.
    PreparedStatement ps = getConnection().prepareStatement(
        "SELECT * FROM auto WHERE (marke=?)");
    // Statement wird erzeugt
    ps.setString(1, "bmw"); // Parameter werden übergeben
    ResultSet rs = ps.executeQuery(); // Statement wird ausgeführt.
    while (rs.next()) {
        System.out.print(rs.getString("marke"));
        System.out.print(" : "+rs.getInt("geschwindigkeit"));
        System.out.println();
    }
```


CallableStatement

- ❖ Das Callable-Statement-Objekt dient zum Aufruf von Stored Prozedures in der Datenbank. Diese Prozeduren können in Java selber oder in einer Erweiterung von SQL wie PL/SQL von ORACLE beschrieben sein. Das CallableStatement-Objekt erbt vom PreparedStatement-Objekt.
- ❖ Man unterscheidet bei den Parameter folgende Parametertypen:
 - ❖ IN-Parameter : Nehmen Werte auf, die weiterverarbeitet werden
 - ❖ OUT-Parameter : Enthalten Rückgabewerte
 - ❖ IN/OUT-Parameter: Nehmen Übergabeparameter auf, deren Wert verändert und an das aufrufende Programm zurückgegeben werden kann.

CallableStatement

```
try {
    dbConnection = getDBConnection();
    dbConnection.setAutoCommit(false);
    CallableStatement callableStatement =
        (CallableStatement)dbConnection.prepareCall("{? = call show_autos()}");
    callableStatement.registerOutParameter(1, Types.OTHER);
    callableStatement.execute();
    System.out.println("callable stament is called!");
    ResultSet results = (ResultSet) callableStatement.getObject(1);
    while (results.next()) {
        System.out.println(results);
    }

} catch (SQLException e) {

    System.out.println(e.getMessage());

} finally {
```

Übung

- ❖ Übung 2
- ❖ Übung 3

ResultSet

- ❖ ist ein Interface, das von den verschiedenen Treibern durch entsprechende Klassen implementiert wird.
- ❖ Ergebnis eines SQL-Select-Statements
- ❖ enthält Tabelle oder einen Teil einer Tabelle.

Standardresultset

- ❖ Resultset verfügt über einen Zeilenzeiger (Cursor), der mit der Methode `next()` eine Zeile weiter transportiert wird.
- ❖ Am Anfang steht dieser Zeilenzeiger vor der Tabelle, sodaß er mit einem ersten `next()`-Aufruf auf die erste Zeile der Tabelle gesetzt wird
- ❖ jede Datenzeile kann nur einmal gelesen werden
- ❖ kein Weg zurück

Auslesen des Standardresultsets

- ❖ MetaDaten in einem ResultSet
- ❖ über Objekt vom Typ ResultSetMetaData(Interface) aufrufbar.
- ❖ Spaltenindex beginnt bei 1

ResultSet mit Metadata

```
public static void printResultSet(ResultSet rs) throws SQLException
{
    ResultSetMetaData rsmd = rs.getMetaData();
    int cols = rsmd.getColumnCount();

    for(int i=1; i<=cols; i++)
        System.out.print(rsmd.getColumnLabel(i)+"\t");

    System.out.println("\n-----");

    while(rs.next())
    {
        // eine zeile ausgeben
        for(int i=1; i<=cols; i++)
            System.out.print(rs.getString(i)+"\t");

        System.out.println();
    }
}
```

ResultSet einfach

```
while (rs.next()) {  
    System.out.print(rs.getString("marke"));  
    | System.out.print(" : "+rs.getInt("geschwindigkeit"));  
    System.out.println();  
}
```

Übung

- ❖ Übung4
- ❖ Übung5
- ❖ Übung6