JDBC – Java Database Connectivity Einführung

Lukas Wais

Codersbay

27. Februar 2020

Inhaltsverzeichnis

Overview

Datenbank APIs ohne JDBC

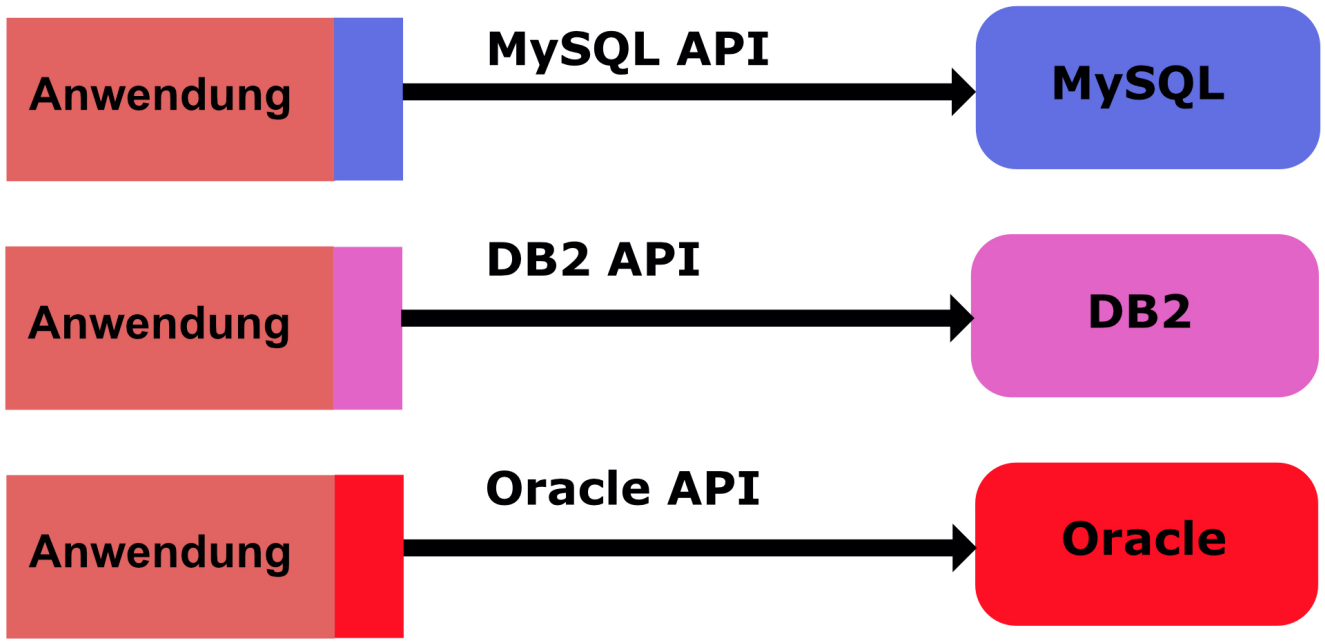

Abbildung: Datenbank APIs ohne JDBC

API . . . Application Programming Interface

Overview

Datenbank APIs ohne JDBC

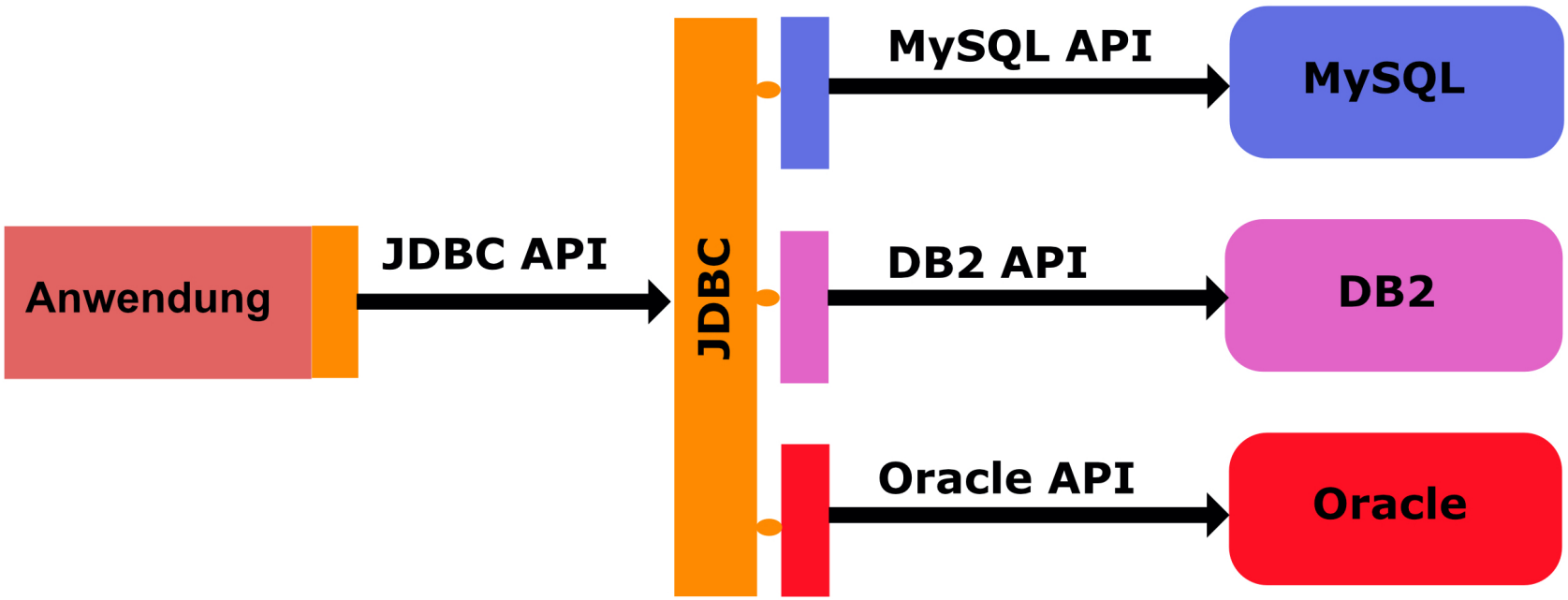

Abbildung: Datenbank APIs mit JDBC

Datenbankzugriffsschnittstelle für Java

- abstrakt und datenbankneutral
- vergleichbar mit ODBC
- Low-Level-API: direkte Nutzung von SQL
- Java-Package java.sql
- entwickelt von Sun Microsystems

Klassen

- **DriverManager:** Einstiegspunkt, Laden von Treibern
- **Connection:** Datenbankverbindung
- **Statement:** Ausführung von Anweisungen über eine Verbindung
- **ResultSet:** verwaltet Ergebnisse einer Anfrage, Zugriff auf einzelne Spalten

Ablauf der Datenbankverbindung

Diagramm

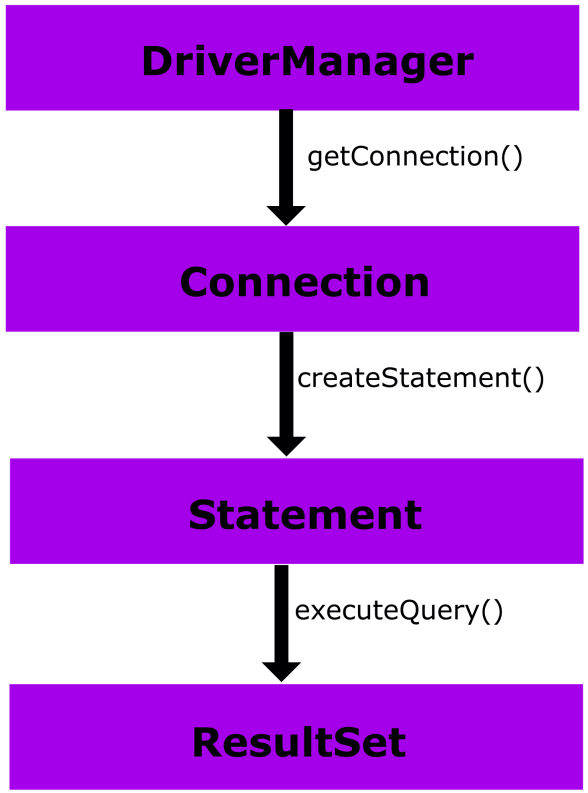

Abbildung: Datenbankverbindung

Ablauf der Datenbankverbindung

Erklärung

- Aufbau einer Verbindung zur DB
  - Angabe der Verbindungsinformationen
  - Auswahl und dynamisches Laden des Treibers
- Senden einer SQL-Anweisung
  - Definition der Anweisung
  - Belegung von Parametern
- Verarbeiten der Anfrageergebnisse
  - Navigation über Ergebnisrelation
  - Zugriff auf Spalten

Ablauf der Datenbankverbindung

Sourcecode

Netbeans

Datentypen

Typabbildung MySQL → Java

MySQL Type Name	Return value of GetColumnTypeName	Return value of GetColumnClassName
BIT(1) (new in MySQL-5.0)	BIT	java.lang.Boolean
BIT(> 1) (new in MySQL-5.0)	BIT	byte[]
TINYINT	TINYINT	java.lang.Boolean if the configuration property tinyInt1isBit is set to true (the default) and the storage size is 1, or java.lang.Integer if not.
BOOL, BOOLEAN	TINYINT	See TINYINT, above as these are aliases for TINYINT(1), currently.
SMALLINT[(M)] [UNSIGNED]	SMALLINT [UNSIGNED]	java.lang.Integer (regardless of whether it is UNSIGNED or not)
MEDIUMINT[(M)] [UNSIGNED]	MEDIUMINT [UNSIGNED]	java.lang.Integer (regardless of whether it is UNSIGNED or not)
INT,INTEGER[(M)] [UNSIGNED]	INTEGER [UNSIGNED]	java.lang.Integer, if UNSIGNED java.lang.Long
BIGINT[(M)] [UNSIGNED]	BIGINT [UNSIGNED]	java.lang.Long, if UNSIGNED java.math.BigInteger
FLOAT[(M,D)]	FLOAT	java.lang.Float
DOUBLE[(M,B)]	DOUBLE	java.lang.Double
DECIMAL[(M[,D])]	DECIMAL	java.math.BigDecimal
DATE	DATE	java.sql.Date
DATETIME	DATETIME	java.sql.Timestamp
TIMESTAMP[(M)]	TIMESTAMP	java.sql.Timestamp
TIME	TIME	java.sql.Time

Abbildung: Auszug Typabbildung

Der Link zur gesamten Liste `https://bit.ly/2uA5quV`

Definition

Entwurfsmuster, Architekturmuster und konkreter Architektur

Während eine **konkrete Architektur** neben der Erfüllung nicht funktionaler Eigenschaften auch **konkreten funktionalen Anforderungen** genügt, stehen bei einer Referenzarchitektur und bei einem **Architekturmuster** bzw. **Entwurfsmuster** die nicht funktionalen Anforderungen wie **Standardisierung, Verständlichkeit, Einfachheit** und **Ausbaufähigkeit** im Vordergrund. [1]

Solche Muster sind nicht auf Java beschränkt, sondern können bei allen objektorientierten Sprachen angewendet werden.

Entwurfsmuster

Definition Entwurfsmuster

In der objektorientierten Softwareentwicklung sind **Entwurfstmuster Klassen in Rollen,** die zusammenarbeiten, um **gemeinsam** eine **bestimmte Aufgabe** zu **lösen**.[1]

Entwurfsmuster vs. Architekturmuster

Unterschied Architekturmuster und Entwurfsmuster

Entwurfsmuster stellen **feinkörnige Muster** dar, während **Architekturmuster grobkörnige Muster** sind. [1]

Einige Beispiele

Entwurfsmuster:

- Visitor Pattern
- Factory Pattern
- Singleton Pattern

Architekturmuster:

- MVC . . . Model View Controller
- Schichtenarchitektur
- Client-Server

Definition

Definition Singleton

Das **Singleton-Muster** soll gewährleisten, dass eine Klasse nur ein einziges Mal instanziert werden kann. [1]

Beispiel

Problem

Es soll eine Klasse geben, von der sichergestellt werden muss, dass nur eine einzige Instanz von ihr existiert. Für das Erzeugen der einzigen Instanz soll die genannte Klasse selbst verantwortlich sein. [1]

Beispiel

Lösung

Ein Singleton Objekt kann nur von der Klasse *Singleton* selbst erzeugt werden. Alle Konstruktoren dieser Klasse werden mit dem Zugriffsmodifier *private* gekennzeichnet, so dass andere Klassen kein Objekt unter Verwendung eines Konstruktors erzeugen können. Objekte, die die Klassen *Singleton* verwenden möchten, erhalten von der Klassenmethode *getInstance()* der Klasse *Singleton* eine Referenz auf das einzig existierende Objekt der Klasse *Singleton* zurück. [1]

UML Diagramm

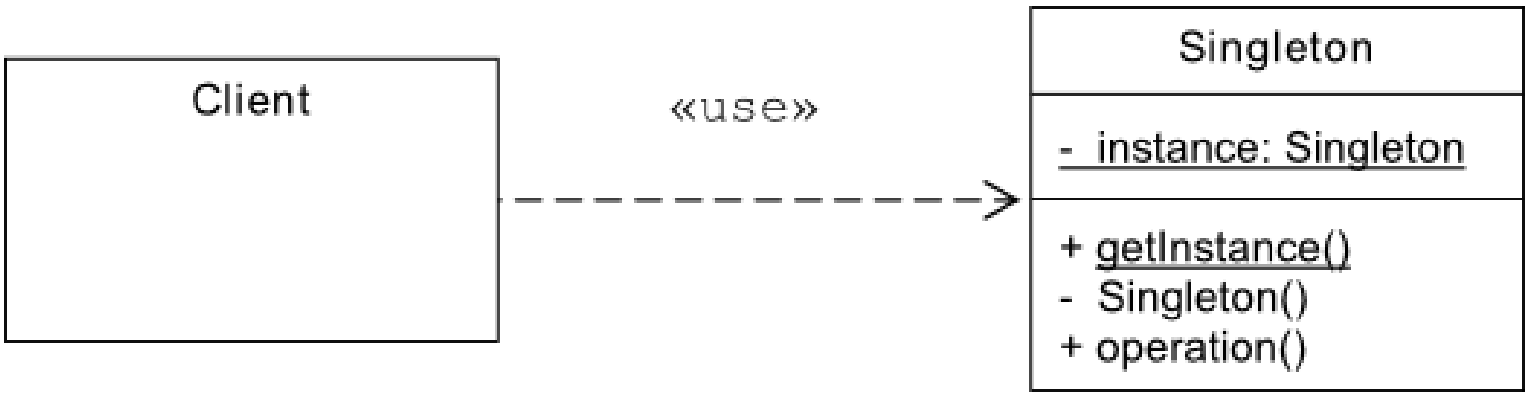

Abbildung: UML Diagramm Singleton

[1]

Sourcecode

```
public class Singleton {

   private static Singleton instance = new Singleton();

   private Singleton(){/*...*/}

   // Get the only object available
   public static Singleton getInstance(){
      return instance;
   }
}
```

Setup der Datenbank

- username: codersbay
- password: codersbay
- Link zu phpmyAdmin
  `https://www.db4free.net/phpMyAdmin/`
- Download Connector/J 8.0.19 für MySQL
  `https://dev.mysql.com/downloads/connector/j/`
- **Link für Java**
  "jdbc:mysql://db4free.net:3306/codersbayworld?
  zeroDateTimeBehavior=CONVERT_TO_NULL"

Verwendung von JDBC

SQL Statements in Java

SQL Statements sollten aus Performancegründen sehr sparsam eingesetzt werden. Nach Möglichkeit das Datenbankmanagementsystem DBMS verwenden.
Cursor, Trigger, PLSQL, Exceptions, ...

Referenzen


J. Goll and M. Dausmann, *Architektur-und Entwurfsmuster der Softwaretechnik*.
Springer, 2013.