STUDIEREN. WISSEN. MACHEN.

HOCHSCHULE DER MEDIEN

Datenbankanbindung

SERVERSEITIGE KONZEPTE

Web Technologie - SS19 | Prof. Dr. Christoph Kunz

JDBC

HOCHSCHULE DER MEDIEN

- Java Database Connectivity
- Datenbankschnittstelle der Java-Plattform zum Zugriff auf Datenbanken verschiedener Hersteller
- Spezielle Ausrichtung auf relationale Datenbanken
- JDBC ist in seiner Funktion als universelle Datenbankschnittstelle vergleichbar mit z. B. ODBC unter Windows
- Jede spezifische Datenbank benötigt eigene Treiber, welche die JDBC-Spezifikation implementieren
- Diese Treiber werden meist vom Hersteller des Datenbank-Systems geliefert

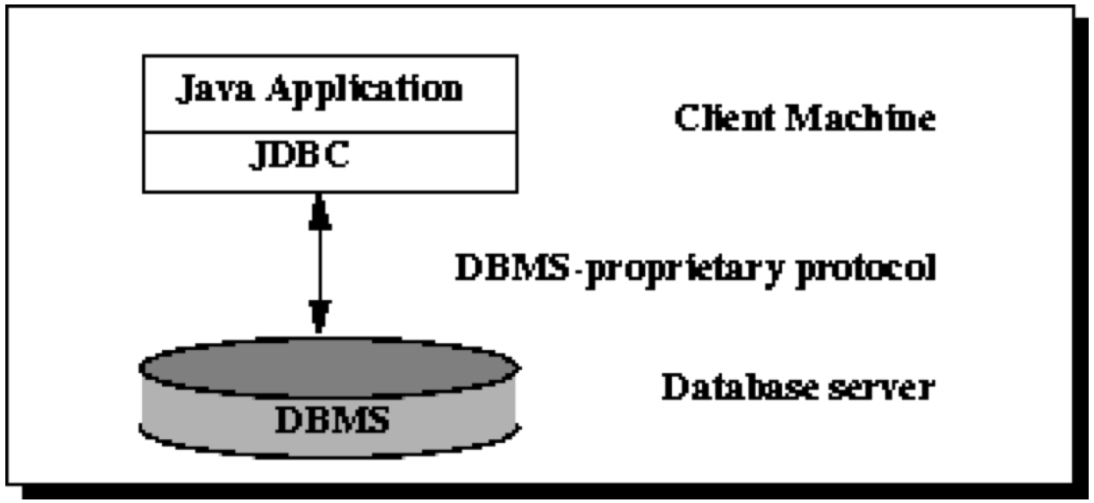
Aufgaben

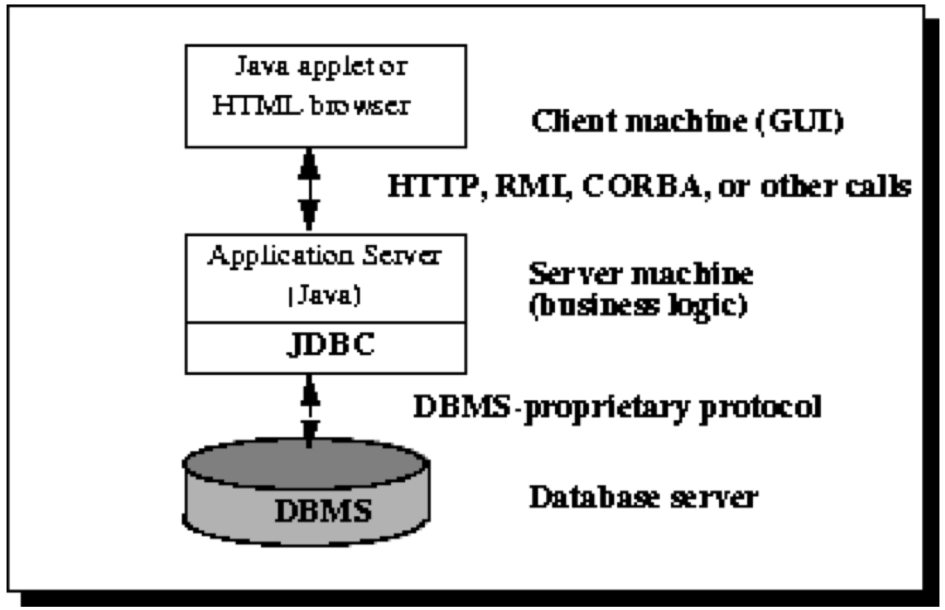
- Verbindung zur Datenbank aufnehmen
- Anfragen und Aktualisierungen zur Datenbank übermitteln
- Ergebnisse einer Datenbankanfrage verarbeiten

```
public void connectToAndQueryDatabase(String username, String password) {

    Connection con = DriverManager.getConnection(
                       "jdbc:myDriver:myDatabase",
                       username,
                       password);

    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");

    while (rs.next()) {
        int x = rs.getInt("a");
        String s = rs.getString("b");
        float f = rs.getFloat("c");
    }
}
```

Komponenten

- JDBC API (Packages `java.sql` und `javax.sql` als Teil der Java Standard Edition)
- JDBC Driver Manager erzeugt mit Hilfe von Datenbanktreibern Objekte, die sich zu einer Datenbank verbinden können.
- (JDBC Test Suite, JDBC-ODBC-Bridge)





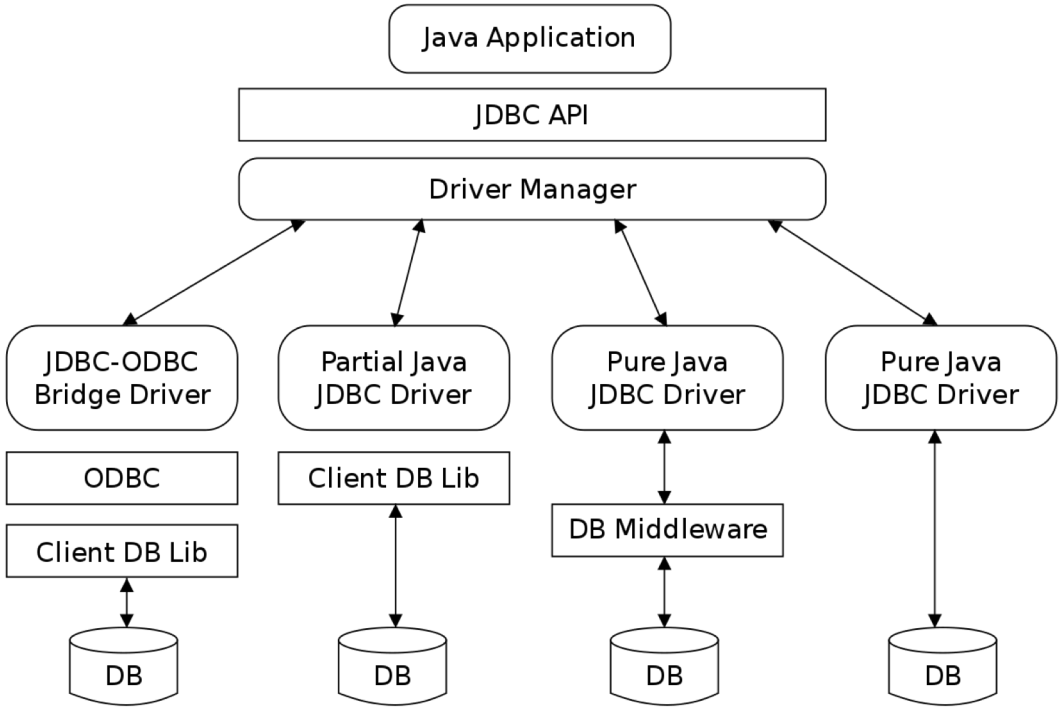
Typen von JDBC Treibern

Typ-1-Treiber:

- kommuniziert ausschließlich über einen JDBC-ODBC-Bridge-Treiber (z.B. Oracle).
- Der JDBC-ODBC-Bridge-Treiber wandelt JDBC- in ODBC-Anfragen um.
- Mit Java 9 wird die Unterstützung für JDBC-Typ-1-Treiber eingestellt.

Typ-2-Treiber:

- kommuniziert über eine plattformspezifische Programmbibliothek auf dem Client mit dem Datenbankserver.



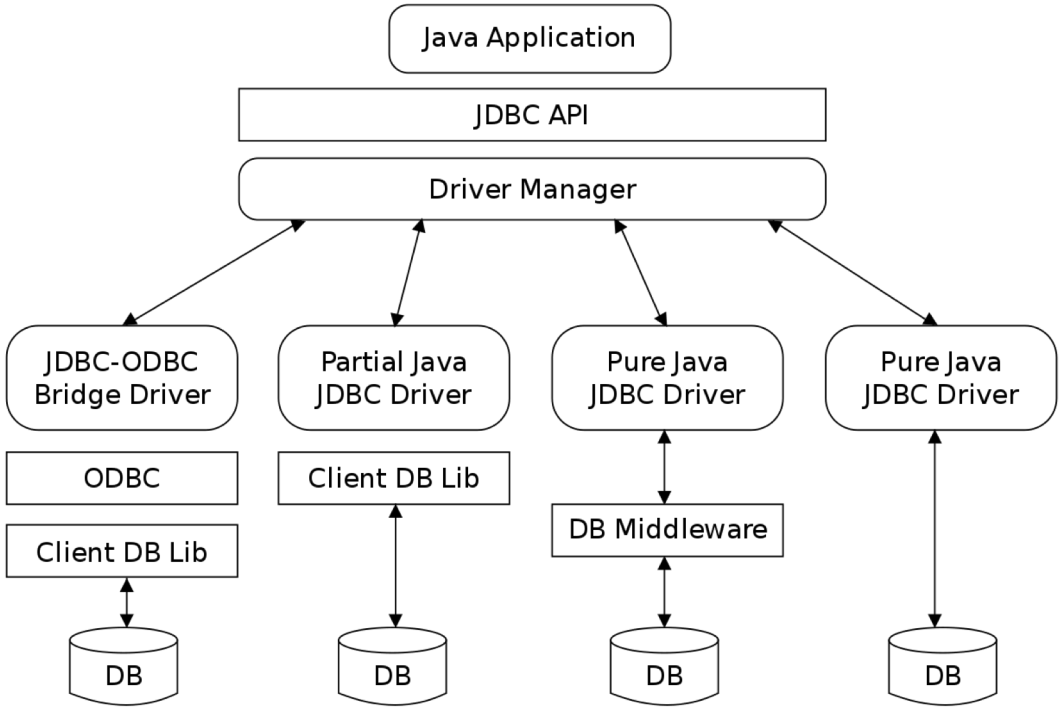
Typen von JDBC Treibern

Typ-3-Treiber

- JDBC-API-Befehle werden in generische DBMS-Befehle übersetzt und (über ein Netzwerkprotokoll) an einen Middleware-Treiber auf einem Anwendungsserver übertragen.
- Anwendungsserver transformiert die Befehle für die spezifischen Datenbankserver und leitet sie weiter.
- Typ-3-Treiber eignen sich sehr gut für Internet-Protokolle im Zusammenhang mit Firewalls.

Typ-4-Treiber

- JDBC-API-Befehle werden direkt in DBMS-Befehle des jeweiligen Datenbankservers übersetzt und (über ein Netzwerkprotokoll) an diesen übertragen.
- Ein Typ-4-Treiber kann schneller als ein Typ-3-Treiber sein, ist aber weniger flexibel.
- Typ-4-Treiber eignen sich gut für Intranet-Lösungen, die schnelle Netzprotokolle nutzen wollen.



Vorbereitung

HOCHSCHULE DER MEDIEN

- Bereitstellung/Zugriff auf ein Datenbank-Management-System
  - z.B. MySQL, Derby Java DB
- Installation des JDBC-Treibers vom Datenbankhersteller
  - z.B. die aktuelle Version von „Connector/J“ von MySQL
- Hinzufügen der jar-Datei zum Classpath

- Manuelle MySQL Installation
  - MySQL Shell
  - MySQL Workbench
- Installation eines xAMP Stacks
  - LAMP: Linux Apache, MySQL, PHP
  - XAMPP Apache, MariaDB, PHP, Perl
  - MAMP (Pro): Mac, Apache, MySQL, PHP
  - MAMP auch für Windows
  - Enthält PHP myAdmin

Verarbeiten von SQL-Statements

HOCHSCHULE DER MEDIEN

- Aufbau einer Verbindung zur Datenbank
- Erzeugen eines Statements
- Ausführen eines Statements
- Verarbeiten des ResultSet-Objekts
- Schließen der Verbindung

Beispiel

```
public Vector<Customer> findAll() {
        // Verbindung Datenbank erhalten
        Connection con = DBConnection.connection();
        try {
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(
                    "SELECT id, firstName, lastName " +
                    "FROM customers " +
                    "ORDER BY lastName");

            // Fuer jeden Eintrag im Suchergebnis wird nun ein Customer-Objekt erstellt.
            Vector<Customer> result = new Vector<Customer>();
            while (rs.next()) {
                int id = rs.getInt("id");
                Customer c = new Customer();
                c.setId(id);
                c.setFirstName(rs.getString("firstName"));
                c.setLastName(rs.getString("lastName"));

                // Hinzufuegen des neuen Objekts zum Ergebnisvektor
                result.addElement(c);
            }
        } catch (SQLException e2) {
            e2.printStackTrace();
        }
      return result;
}
```

Herstellen einer Verbindung

HOCHSCHULE DER MEDIEN

- Verwendung des DriverManager. Dieser verbindet eine Anwendung zu einer Datenquelle, die durch eine Datenbank-URL vorgegeben ist. (Alternativ: Verwendung des DataSource Interface)

```
DriverManager.getConnection("jdbc:derby:banking")
```

```
DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/banking")
```

- Ergebnis des Aufrufs ist ein Objekt vom Typ Connection, das die Verbindung zur Datenbank repräsentiert.

Aufbau der Verbindungs-URL für Java DB

```
jdbc:derby:[subsubprotocol:][databaseName]
    [;attribute=value]*
```

- `subsubprotocol` gibt an, wo Java nach der Datenbank suchen soll (Server, Verzeichnis, Hauptspeicher, Classpath, Jar-Datei)
- `databaseName` ist der Name der Datenbank
- `attribute=value` ist eine optionale durch Semikolons getrennte Liste von Attributen, mit deren Hilfe
  - die Datenbank erzeugt werden kann.
  - die Datenbank verschlüsselt werden kann.
  - Verzeichnisse zum Speichern von Protokollen spezifiziert werden können.
  - der Name und das Passwort eines Nutzers angegeben werden kann.

Aufbau der Verbindungs-URL für MySQL

HOCHSCHULE DER MEDIEN

```
jdbc:mysql://[host][,failoverhost...]
    [:port]/[database]
    [?propertyName1][=propertyValue1]
    [&propertyName2][=propertyValue2]...
```

- `host:port` ist der Rechnername und (optional) die Portnumber des Rechners, auf dem die Datenbank läuft.
- `database` ist der Name der Datenbank
- `failover` ist der Name der Ausfalldatenbank
- `propertyName=propertyValue` ist eine optionale, durch „&“ getrennte Liste von Eigenschaften.

```
jdbc:mysql://localhost/banking
```

Behandlung von SQL-Ausnahmen

HOCHSCHULE DER MEDIEN

- SQL-Ausnahmeobjekte beinhalten spezifische Informationen über Natur und Ursache einer Ausnahme beim Zugriff auf die Datenbank.
- `getMessage`: ein String-Objekt, das die Fehlerbeschreibung enthält.
- `getSQLState`: einen durch ISO/ANSI standardisierten Code in Form eines Strings, der aus 5 alphanumerischen Zeichen besteht.
- `getErrorCode`: Implementationsspezifischer Fehlercode als Integer-Wert.
- `getCause`: ein Ausnahmeobjekt, das Ursache diese Ausnahmeobjektes ist, falls die Ausnahme „weitergeworfen“ wurde, oder `null`.
- `getNextException`: Kette von Ausnahmeobjekten, falls mehr als ein Fehler aufgetreten ist.

Behandlung von SQL-Warnungen

HOCHSCHULE DER MEDIEN

- `SQLWarning` ist Subklasse von `SQLException`
- Sie halten die Ausführung nicht an, sondern versehen bestimmte Objekte (`Connection`, `Statement`, `ResultSet`) mit „Warninformationen“, falls beim Datenbankzugriff etwas Ungewöhnliches passiert.
- Auf die Warnungen kann mit der Methode `getWarnings` (und dann `getNextWarning`) zugegriffen werden.

```
public static void printStatementWarnings(Statement stmt) throws SQLException {
    SQLWarning warning = stmt.getWarnings();

    while (warning != null) {
        System.out.println("Message: " + warning.getMessage());
        System.out.println("SQLState: " + warning.getSQLState());
        System.out.print("Vendor error code: ");
        System.out.println(warning.getErrorCode());
        System.out.println("");
        warning = warning.getNextWarning();
    }
}
```

Tabellen einrichten

HOCHSCHULE DER MEDIEN

- Nach Aufbau der Verbindung:
  - Erzeugen eines `Statement`-Objekts
  - Ausführen von SQL-Code zum Erzeugen der Tabelle mit Hilfe von `executeUpdate`

```
public void createTable() {
    String createString = "create Table accounts"
     + "(id int(11) NOT NULL default '0', "
     + "owner int(11) NOT NULL default '0', "
     + "balance float NOT NULL default '0', "
     + "PRIMARY KEY (id)) "
     + "ENGINE=MyISAM DEFAULT CHARSET=latin1";
    Connection con = DBConnection.connection();
    Statement stmt;
    try {
        stmt = con.createStatement();
        stmt.executeUpdate(createString);
    } catch (SQLException e) {
    }
}
```

Tabelle befüllen

HOCHSCHULE DER MEDIEN

- Nach Aufbau der Verbindung:
  - Erzeugen eines `Statement`-Objekts
  - Ausführen von SQL-Code zum Befüllen der Tabelle mit Hilfe von `executeUpdate`

```
public void fillTable() {
    String sqlString = "INSERT INTO accounts (id,owner,balance) VALUES "
            + "(1,1,300.23), (2,7,1000.94), (3,5,978278), "
            + "(4,6,215652), (5,6,121232), (6,3,32), "
            + "(7,10,10105.8), (8,6,120), (9,6,3e+006)";
    Connection con = DBConnection.connection();
    Statement stmt;
    try {
        stmt = con.createStatement();
        stmt.executeUpdate(sqlString);
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}
```

Zugriff und Modifikation von Werten der Resultate

HOCHSCHULE DER MEDIEN

› `executeQuery` für Abfragen

```
public Vector<Account> findAll() {
   Connection con = DBConnection.connection();
   Vector<Account> result = new Vector<Account>();
   try {
     Statement stmt = con.createStatement();
     ResultSet rs = stmt.executeQuery("SELECT id, owner, balance FROM accounts " + " ORDER BY id");
     while (rs.next()) {
         int id = rs.getInt("id");
         Account a = new Account();
         a.setId(id);
         a.setOwner(CustomerMapper.customerMapper().
              findByKey(rs.getInt("owner")));
         a.setBalance(rs.getFloat("balance"));

       result.addElement(a);
      }
   } catch (SQLException e2) {
      e2.printStackTrace();
   }
   return result;
}
```

ResultSet

HOCHSCHULE DER MEDIEN

- Ein `ResultSet`-Objekt ist eine Datentabelle, die das Ergebnis einer Anfrage repräsentiert.
- Zugriff auf die Elemente (Tupel, Datensätze) erfolgt über einen sog. „Cursor“. Dieser zeigt immer auf eine bestimmte Zeile der Ergebnistabelle.
- Anfänglich zeigt der Cursor vor(!) den ersten Datensatz.
- Die Methode `next` bewegt den Cursor zum nächsten Datensatz und ergibt `false`, falls sie nach der letztem positioniert wird.

Zugriff auf Attributwerte von Datensätzen

HOCHSCHULE DER MEDIEN

- Zugriffsmethoden sind im ResultSet-Interface festgelegt.
- Nutzung von Attributnamen oder den Indizes der Spalten beginnend bei 1.

```
…
    ResultSet rs
      = stmt.executeQuery(
           "SELECT id, owner, balance FROM accounts " + " ORDER BY id");
    while (rs.next()) {
        int id = rs.getInt("id");
        Account a = new Account();
        a.setId(id);
        a.setOwner(CustomerMapper.customerMapper().findByKey(rs.getInt("owner")));
        a.setBalance(rs.getFloat("balance"));
        result.addElement(a);
     }
 …
```

Zugriffsmethoden von ResultSet

HOCHSCHULE DER MEDIEN

Auf Daten

› `getInt, getBoolean, getByte, getDate, getDouble, getFloat, getLong, getShort, getString, getTime, getTimtestamp, getURL, etc.`

Um den Cursor zu bewegen

› `next, previous, first, last, beforeFirst, afterLast, relative, absolute, etc.`

Vorbereitete (Prepared) Statements

HOCHSCHULE DER MEDIEN

- macht häufig ausgeführte SQL-Anweisungen effizienter
- wird vom DBMS vorkompiliert
- kann mit und ggf. ohne Parameter verwendet werden.

```
PreparedStatement updateBalance
    = con.prepareStatement("UPDATE accounts SET balance=? WHERE id=?");

updateBalance.setFloat(1, a.getBalance());
updateBalance.setInt(2, a.getId();
updateBalance.executeUpdate();
```

Transaktionen

HOCHSCHULE DER MEDIEN

- „eine Anzahl von Statements, die als Einheit ausgeführt werden, so dass entweder alle oder keine ausgeführt werden.“
- zur Vermeidung temporärer Inkonsistenzen
- Normalerweise werden einzelne Statements sofort ausgeführt („Auto-Commit-Modus“)

Nach

```
con.setAutoCommit(false)
```

wird keine Statement mehr ausgeführt, bis explizit die Methode `commit` augerufen wird.

Weiter JDBC Fähigkeiten

HOCHSCHULE DER MEDIEN

- Sog. RowSet-Objects
- Komplexere Datentypen
- Große Binärbjekte (BLOBs)
- Stored Procedures

Data Mapping

HOCHSCHULE DER MEDIEN

- … beschreibt die Abbildung von Objekten in Datensätze (Tupel) und umgekehrt.
- In Java werden Anwendungsdaten (z.B. Banken, Kunden und Konten) als Objekte entsprechender Klassen dargestellt.
- Die Beziehungen zwischen den Objekten (z.B. „Kontoinhaber“) sind als Attributwerte realisiert.
- In relationalen Datenbanken werden Anwendungsdaten (z.B. Banken, Kunden und Konten) als Einträge/Datensätze in Tabellen abgelegt.
- Die Beziehungen zwischen den Datensätzen werden über Fremdschlüssel realisiert.
- Die Umformung zwischen Objekten und Datensätzen erfolgt am besten in sog. Mapper-Klassen.

Mapping: Tupel -> Objekt (findByKey)

```
public Customer findByKey(int id) {
     // DB-Verbindung holen
     Connection con = DBConnection.connection();
     try {
        // Leeres SQL-Statement (JDBC) anlegen
        Statement stmt = con.createStatement();
        // Statement ausfuellen und als Query an die DB schicken
        ResultSet rs = stmt.executeQuery(
               "SELECT id, firstName, lastName FROM customers " +
               "WHERE id=" + id + " ORDER BY lastName");
        /* Da id Primaerschluessel ist, kann max. nur ein Tupel zurueckgegeben werden.
         * Pruefe, ob ein Ergebnis vorliegt.
         */
        if (rs.next()) {
           // Ergebnis-Tupel in Objekt umwandeln
           Customer c = new Customer();
           c.setId(rs.getInt("id"));
           c.setFirstName(rs.getString("firstName"));
           c.setLastName(rs.getString("lastName"));
           return c;
        }
     } catch (SQLException e2) {
         e2.printStackTrace();
     }
     return null;
}
```

Das Tupel in der Ergebnistabelle wird in ein Objekt mit entsprechenden Attributen überführt.

Mapping: Objekt -> Tupel (insert)

```
public void insert(Customer c) {
   Connection con = DBConnection.connection();
   try {
       Statement stmt = con.createStatement();
       /* Zunaechst schauen wir nach, welches der momentan hoechste
        * Primaerschluesselwert ist.
        */
       ResultSet rs = stmt.executeQuery("SELECT MAX(id) AS maxid " + "FROM customers");
       // Wenn wir etwas zurueckerhalten, kann dies nur einzeilig sein
       if (rs.next()) {
           // erhaelt den bisher maximalen, nun um 1 inkrementierten Primaerschluessel.
           c.setId(rs.getInt("maxid") + 1);
           stmt = con.createStatement();

           // Jetzt erst erfolgt die tatsaechliche Einfuegeoperation
           stmt.executeUpdate("INSERT INTO customers (id, firstName, lastName) " +
                   "VALUES (" +
                   c.getId() + ",'" +
                   c.getFirstName() + "','" +
                   c.getLastName() + "')");
       }
   } catch (SQLException e2) {
       e2.printStackTrace();
   }
 }
```

Das einzufügende Objekt wird in ein Tupel mit entsprechenden Attributen überführt.

API des CustomerMapper

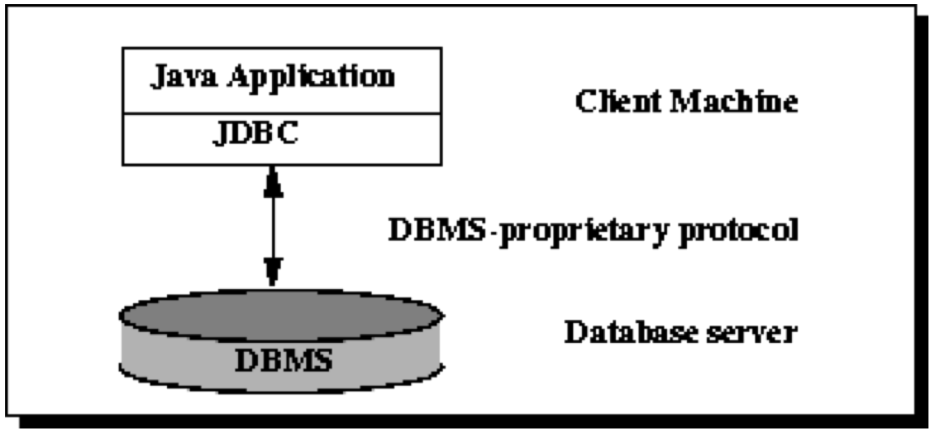
HOCHSCHULE DER MEDIEN

Methodenname	Rückgabewert
findByKey(int id)	Customer
findAll()	Vector<Customer>
findByLastName(String name)	Vector<Customer>
insert(Customer c)	void
update(Customer c)	void
delete(Customer c)	void
getAccountsOf(Customer c)	Vector<Account>

Verwendung der Mapper

HOCHSCHULE DER MEDIEN

- … am Beispiel einer 2-Schichten-Architektur



- Anwendungslogik als Methodenaufrufe in einer zentralen Klasse, hier: `BankAdministration`
- Methoden zum Erzeugen, Löschen und Ändern der zentralen Anwendungsobjekte (Kunden, Konten)
- Mapper sind in jeder Methode verwendbar.

```
public class BankAdministration {
    private CustomerMapper cMapper;
    private AccountMapper aMapper;
    ...
}
```

Erzeugen und Abspeichern eines Anwendungsobjekts

HOCHSCHULE DER MEDIEN

- Erzeugen des Objekts als Instanz der Klasse `Customer`
- Aufruf der Mapper-Methode zur Abspeicherung in der Datenbank.

```
public Customer createCustomer(String first, String last) {
        Customer c = new Customer();
        c.setFirstName(first);
        c.setLastName(last);
        /*
         * Setzen einer vorlaeufigen Kundennr. Der insert-Aufruf
         * liefert dann ein
         * Objekt, dessen Nummer mit der Datenbank konsistent ist.
         */
        c.setId(1);

        // Objekt in der DB speichern.
        return cMapper.insert(c);
}
```

Zugriff auf ein Anwendungsobjekt

HOCHSCHULE DER MEDIEN

› Aufruf der Mapper-Methode

```
public Vector<Customer> getCustomerByName(String lastName) {

        return this.cMapper.findByLastName(lastName);
}
```

```
public Vector<Customer> getAllCustomers() {

        return this.cMapper.findAll();
}
```

API der BankAdministration

HOCHSCHULE DER MEDIEN

Methodenname	Rückgabewert
createAccountFor(Customer)	Account
createCustomer(String, String)	Customer
deleteAccount(Account)	void
deleteCustomer(Customer)	void
modifyAccount(Account, float)	float
modifyCustomer(Customer, String, String)	Customer
getAccountsOf(Customer)	Vector<Account>
getAllCustomers()	Vector<Customer>
getAllAccounts()	Vector<Account>
getCustomerById(int)	Customer
getCustomerByName(String)	Vector<Customer>