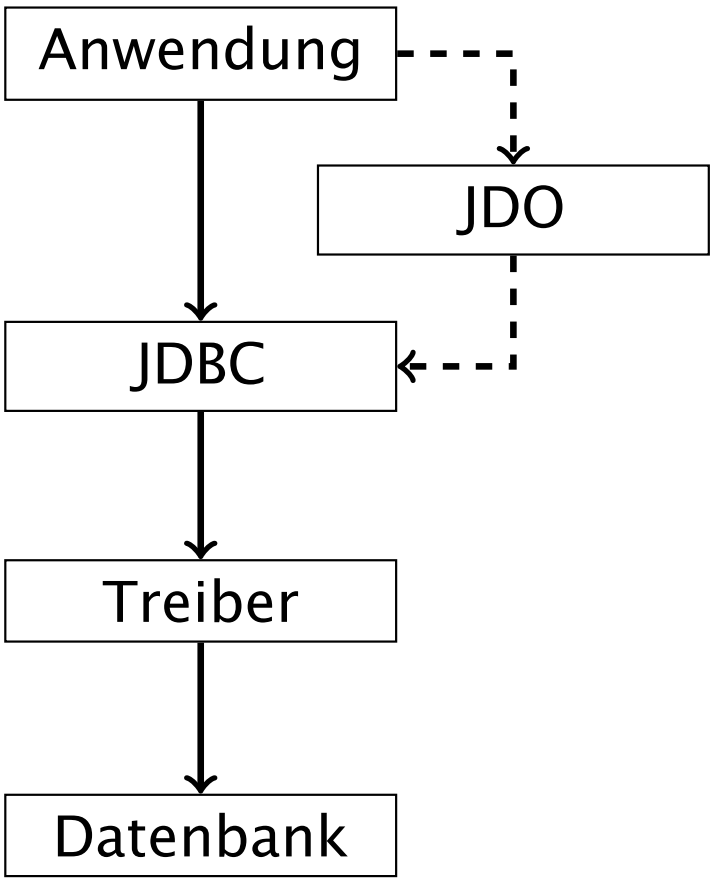
JDBC

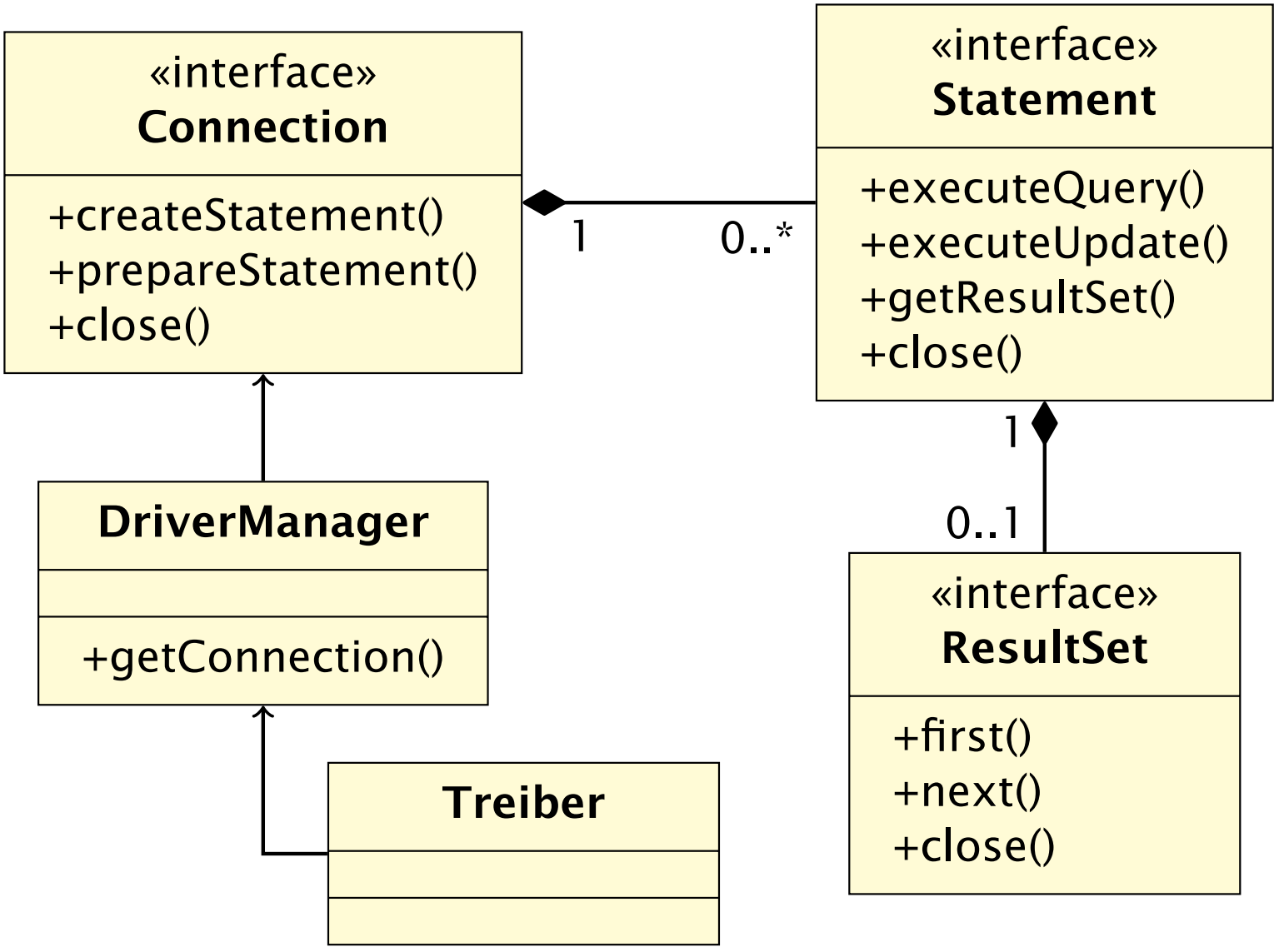
- Java Database Connectivity (JDBC) ist eine SQL-basierte von Datenbankanbietern unabhängige Schnittstelle zwischen Java und relationalen Datenbanken.
- Unterschiedliche SQL-Dialekte muss der Programmierer ausgleichen.
- Es gibt abstraktere Java-Datenbank-APIs (z.B. JDO, Hibernate).
- Diese setzen auf JDBC auf.

Architektur

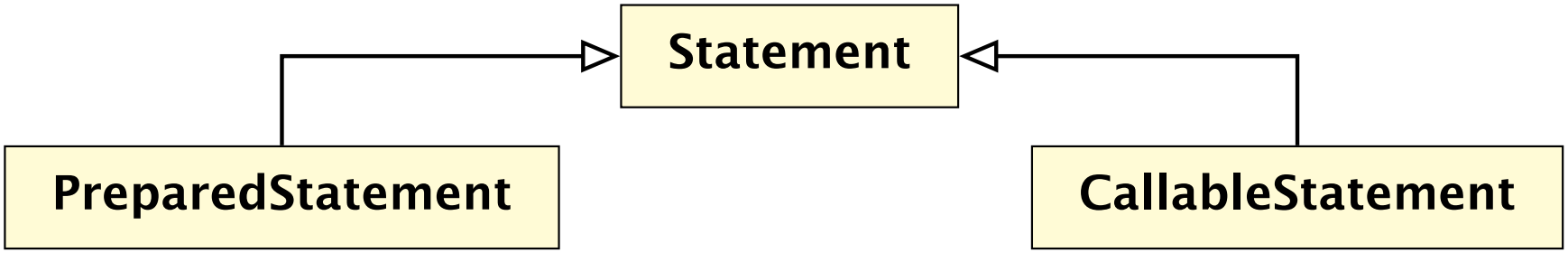


- Anwendungen benutzen das systemunabhängige API JDBC.
- (Alternative: Sie nutzen ein API wie *Java Data Objects* oder ein Framework wie *Hibernate*.)
- JDBC greift auf einen Datenbank-spezifischen Treiber zu.
- Der Treiber interagiert mit dem Datenbanksystem.

Architektur



Architektur (Detail)



- Ein gewöhnliches `Statement` wird mit einem unveränderlichen SQL-Befehl aufgerufen.
- `CallableStatement`s und `PreparedStatement`s nehmen Parameter entgegen und werden meist mehrmals aufgerufen.
- `CallableStatement`s rufen *Stored Procedures* auf.
- `PreparedStatement`s verwenden Sie, wenn ein gleichartiger Befehl mehrmals benötigt wird und wenn Sie Benutzereingaben an die Datenbank durchreichen müssen (Schutz vor *SQL Injection*).

Treiber-Typen

Es gibt vier Typen von JDBC-Treibern:

1. (ODBC-Brückentreiber): Ruft über die ODBC-Schnittstelle einen weiteren Datenbanktreiber auf.
2. benutzt Datenbank-Client-Software auf dem Computer, auf dem die Java-Software läuft.
3. benötigt auf dem Server eine Middleware.
4. spricht direkt mit der Datenbank.

- Vermeiden Sie Typ1-Treiber, sie sind eine Notlösung.
- Typ2-Treiber sind nur im Intranet sinnvoll, wenn die Client-Software ohnehin installiert ist.

Ablauf einer Datenbanksitzung

1. Laden des Treibers mit `Class.forName()`
   (Entfällt seit Java 2)
2. Erstellen einer Connection mit dem DriverManager
3. Erzeugen von Statements
4. Nutzen von ResultSets
5. Schließen aller offenen Objekte

Das ist eine klassische Sitzung. Wir befassen uns nicht mit *DataSource Objects*.

Impedanz-Mismatch

Objektorientierung	*Datenbank*
—	Primärschlüssel
einseitige Beziehung	zweiseitige Beziehung
Eigenschaften und Verhalten	nur Eigenschaften
Vererbung	—
String	VARCHAR

Abbilden von Vererbung

Es gibt drei Methoden; keine ist immer optimal.

1. Eine einzige Tabelle mit Spalten für alle Attribute der Eltern- und aller Kindklassen.
2. Eine Tabelle für die Elternklasse und für jede Kindklasse eine Tabelle für die zusätzlichen Attribute.
3. Eine Tabelle für jede Klasse, Vererbung ist nicht abgebildet.

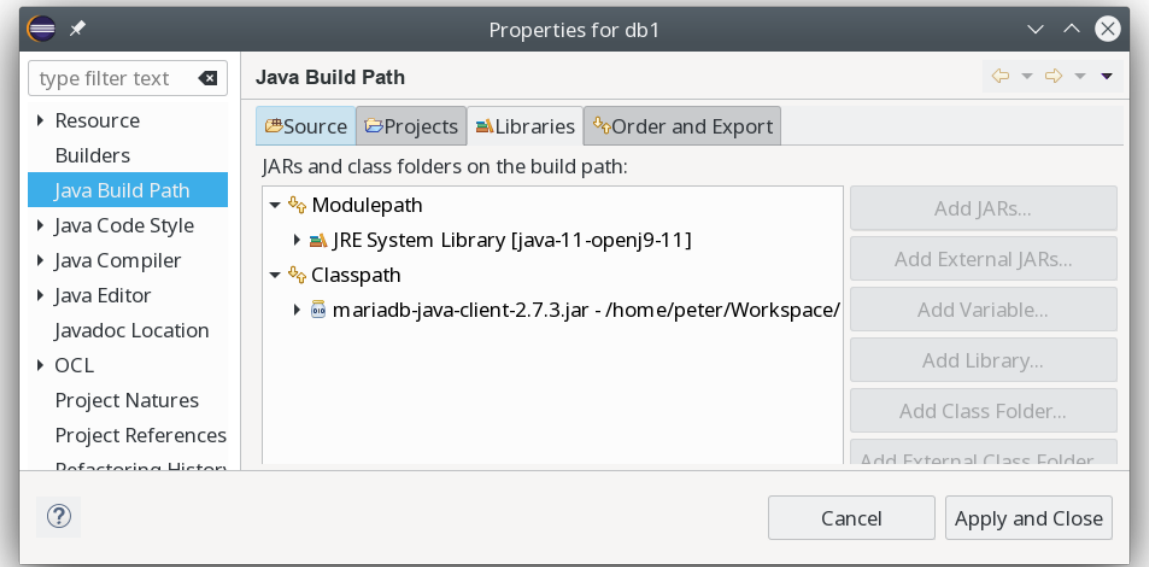
Alternative: Objektorientierte Datenbanken?

Ausschnitt aus einer Werbeanzeige für InterSystems Cache Datenbank, 2004

- Objektorientierte Datenbanken galten in den 2000er Jahren als Nachfolger der relationalen Systeme.
- Sie haben sich nicht durchgesetzt.

Installation

1. Datenbank installieren und konfigurieren.
2. Einige Datenbanken benötigen zusätzlich Client-Software auf dem Gerät, auf dem das Java-Programm läuft.
3. JDBC-Treiber zur Datenbank herunterladen und in Classpath einbinden.
   In Eclipse geht das über Project/Properties/Java Build Path/Libraries.



Handhabung

- Alle JDBC-Aufrufe können Exceptions werfen:
  Binden Sie alle JDBC-Aufrufe in try/catch ein.
- Alle SQL-Objekte (Connection, Statement, ResultSet)
  müssen nach Gebrauch geschlossen werden (`close()`).
- Alle SQL-Objekte sind `AutoClosable`, daher funktioniert
  `try with resources`.

Verbinden (1)

- Zur Verbindung mit der Datenbank wird eine URL benötigt:

```
final String dbsystem = "mariadb";
final String computer = "localhost";
final String database = "BLUTBANK";
final String port = "3306";
String connectionString = "jdbc:" + dbsystem +
"://" + computer + ":" + port+ "/" + database;
```

- Selbstverständlich kann die URL auch komplett angegeben werden:

```
String connectionString =
"jdbc:mariadb://localhost:3306/BLUTBANK";
```

Verbinden (2)

- Benutzerdaten (Name und Passwort) und Datenbankeinstellungen kommen in ein Properties-Hash:
```
Properties benutzerProps = new Properties();
benutzerProps.put("user", "dracula");
connectionProps.put(password", "blutsauger");
```
- In älteren Java-Versionen muss der Datenbank-Treiber vorab geladen werden:
```
Class.forName("org.mariadb.jdbc.Driver");
```

Verbinden (3)

- Dem DriverManager ConnectionString und Properties übergeben.
- Alle Datenbankfunktionen können eine Exception werfen, also try/catch verwenden.
- Eine gründliche Fehlerdiagnose ist sinnvoll.

```
Connection connection;
try {
  connection = DriverManager.getConnection(
     connectionString, connectionProps);
} catch(Exception e) {
// ...
}
```

Trennen (1)

- Verbinden ist aufwändig. Sie wollen es vermutlich nur einmal pro Programmlauf machen.
- Damit eine Datenbankverbindung beim Programmende automatisch geschlossen wird, wird ein Exit-Handler geschrieben.
- Exit-Handler müssen als Thread formuliert werden.
- Dem Exit-Handler wird die Datenbankverbindung übergeben.

Trennen (2)

```
// Exit-Handler definieren
class Datenbanktrenner extends Thread {
  private Connection verbindung;
  public void run() {
    verbindung.close();
  }
  public void setVerbindung(Connection v) {
    verbindung = v;
  }
}
// Im Hauptprogramm, wenn die Verbindung steht:
Datenbanktrenner trenner = new Datenbanktrenner();
trenner.setVerbindung(connection);
Runtime.getRuntime().addShutdownHook(trenner);
```

Datendefinition spontan

```
final String befehl = "delete from BENUTZER";
try {
  statement.executeUpdate(befehl);
} catch(SQLException e) {
  ...
}
```

- Verwenden Sie für DML-[1] und DDL-Befehle `executeUpdate()`.
- Sie erhalten bei DML eine Zahl zurück, die die Anzahl der betroffenen Zeilen angibt.
- DDL-Befehle liefern immer 0 zurück.

[1] DML ist Datenmanipulation: INSERT, UPDATE und DELETE. DDL ist Datendefinition: CREATE, DROP ...

Datendefinition prepared

- Verwenden Sie ein `PreparedStatement` für die Ausführung mehrerer ähnlicher Befehle (Performance).
- Verwenden Sie ein `PreparedStatement` für die Ausführung von Befehlen, die Benutzereingaben enthalten (Sicherheit).

```
final String befehl = "INSERT INTO BENUTZER VALUES (?, ?)";
final String[] namen = { "Fred", "Sabine", "Erika"};
try (PreparedStatement pstmt =
  connection.prepareStatement(befehl)) {
  for(int i = 0; namen.length; i++) {
    pstmt.setInt(1, i + 1);
    pstmt.setString(2, namen[i]);
    pstmt.executeUpdate();
  }
  pstmt.close();
}catch(SQLException e) {

  ...
```

Abfrage spontan

- Rufen Sie auf einem Statement die Methode `ExecuteQuery()` auf.
- Nehmen Sie das ResultSet direkt aus diesem Aufruf entgegen oder holen Sie es mit `getResultSet()`.

```
final String abfrage = "SELECT Nummer, Name from KUNDE";
try (
    Statement stmt = connection.createStatement();
    ResultSet rs = stmt.executeQuery(abfrage);
) {
  while (rs.next()) {
    System.out.print(rs.getInt(1));
    System.out.println("\t"+ rs.getString(2));
  }
} catch(SQLException e) {
```

Abfrage prepared

Ablauf

1. prepare
2. set Parameter
3. execute
4. Daten holen
5. weiter bei 2 oder close

- Einzelne Datenfelder werden im SQL-String durch ? ersetzt.
- Ihnen werden mit `set...(`*Parameternummer*`,` *Wert*`)` Werte zugewiesen.
- Es gibt `set...()` Methoden für jeden SQL-Datentyp: `setBoolean()`, `setDouble()`, `setInt()`, `setString()` ...
- Die Parameternummern beginnen bei 1 (für das erste Fragezeichen).

Abfrage prepared

```
final String abfrage =
  "SELECT Nummer, Name from KUNDE WHERE Name LIKE ?";
try (
    PreparedStatement stmt =
      connection.prepareStatement(abfrage);
) {
  stmt.setString(1, "F%");
  ResultSet rs = stmt.executeQuery();
  while (rs.next()) {
    ...
  }
  rs.close();
} catch(SQLException e) {
```

Ergebnisse abholen

- Das ResultSet setzt *vor* dem ersten Datensatz auf.
- Zur nächsten Zeile gelangen Sie mit `next()`.
- Die Anzahl der Ergebniszeilen ist nicht immer bekannt.
- Wenn Ihnen Spaltenzahl, Datentypen usw. nicht vorab bekannt sind, können Sie mit `getMetaData()` Informationen erhalten.
- Ob Sie »zurückspulen« können, hängt vom Datenbanksystem ab. Verlassen Sie sich nicht darauf, meist ist es nicht erlaubt.
- Schließen Sie das ResultSet, wenn Sie fertig sind.

Ergebnisse abholen

- Es gibt `get...()`-Funktionen für jeden Java-Basisdatentyp.
- Es gibt `get...()`-Funktionen für Spaltennummern und Spaltennamen.
- Spaltennummern werden ab 1 gezählt.

```
try (ResultSet rs = stmt.executeQuery(abfrage);) {
  while (rs.next()) {
    nummer = rs.getInt(1));
    name = rs.getString(2));
    ...
```

Null-Werte

- Verwenden Sie das Schlüsselwort `NULL` innerhalb von SQL-Statements wie üblich.
- In *PreparedStatements* setzen Sie einen Parameter mit `setNull()` auf null. Der Datentyp muss mit angegeben werden.
  ```
  pstmt.setNull(2, Types.VARCHAR));
  ```
- In *ResultSets* können Sie *nach* einem get-Aufruf mit `wasNull()` nachsehen, ob der Wert null war.
  ```
  ort = rs.getString("ORT"));
  if(rs.wasNull())
    System.out.println("Ort ist unbekannt.");
  ```

Fehler auswerten

- Nicht jeder SQL-Fehler ist schlimm.
  Beispiel: Eine Tabelle löschen wollen, die schon gelöscht ist.
- SQLExceptions enthalten

message	den Fehlertext,
SQLState	den genormten Fehlercode (String),
errorCode	einen systemabhängigen Code (int)

- SQLExceptions sollten ausgewertet werden.
- *Warnings* werfen keine Exception. Sie sollten jedoch auch beachtet werden. Beispiel: Ein String wird in ein zu kleines VARCHAR-Feld gepackt.
- Testen Sie, ob Ihr Datenbanksystem SQLState und errorCode unterstützt.

Transaktionen

- JDBC verwendet normalerweise autocommit, das heißt: Jedes Statement wird sofort bestätigt.
- Wenn Transaktionen verwendet werden sollen, muss man sie einschalten.

```
connection.setAutoCommit(false);
// mache irgendwas
connection.commit(); // oder rollback()
connection.setAutoCommit(true);
```