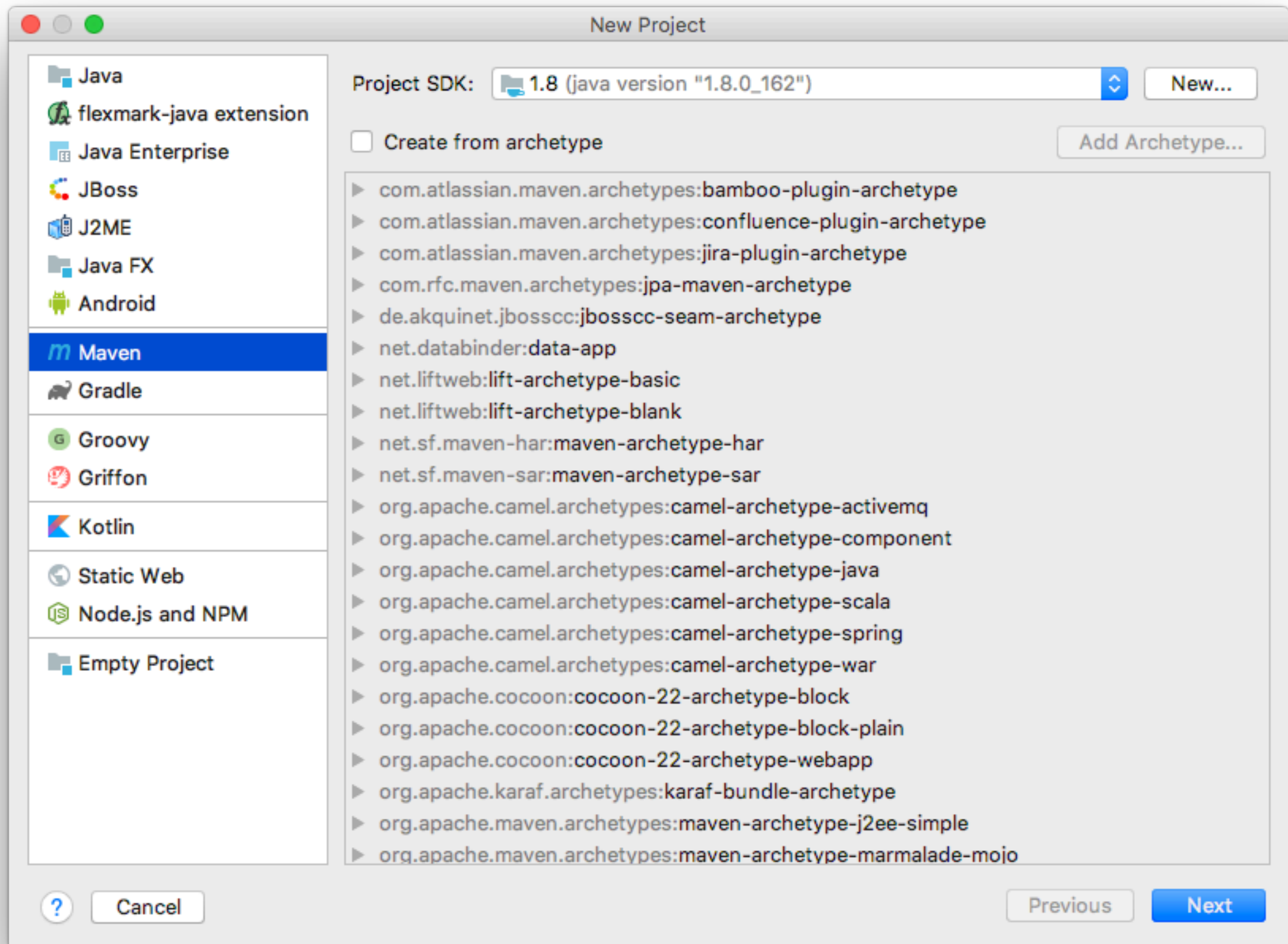


JDBC

<https://github.com/htl-leonding/VehicleJdbc>

<http://tutorials.jenkov.com/jdbc/index.html>

<http://www.tutorialspoint.com/jdbc/>




New Project

GroupId ☒ Inherit

ArtifactId

Version ☒ Inherit



New Project

Project name:

Project location: ...

► More Settings

Projekt erstellen

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>at.htl</groupId>
  <artifactId>vehicle</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.apache.derby</groupId>
      <artifactId>derbyclient</artifactId>
      <version>10.14.2.0</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

Hier könnte man auch 10 oder 11 eintragen. Einstweilen wird 1.8 empfohlen

Datenbank

- Als Datenbank verwenden wir die frühere JavaDb, die korrekterweise **Apache Derby** heißt.
https://db.apache.org/derby/derby_downloads.html
- Man könnte auch andere Datenbanken verwenden, wie zB **H2** - ebenfalls eine Desktop-Datenbank mit kleinem Fußabdruck <http://www.h2database.com/html/download.html>
- Die DerbyDb war bis Java 8 im JDK inkludiert. Ab Java 9 ist sie nicht mehr enthalten.

bis Java 8
(Jdk1.8.0)

Datenbank starten

man öffnet in der IDE ein Terminal - somit befindet man sich im root-Verzeichnis des Projekts. Dort legt man ein Verzeichnis db an

In diesem Verzeichnis werden die DB-Files gespeichert.

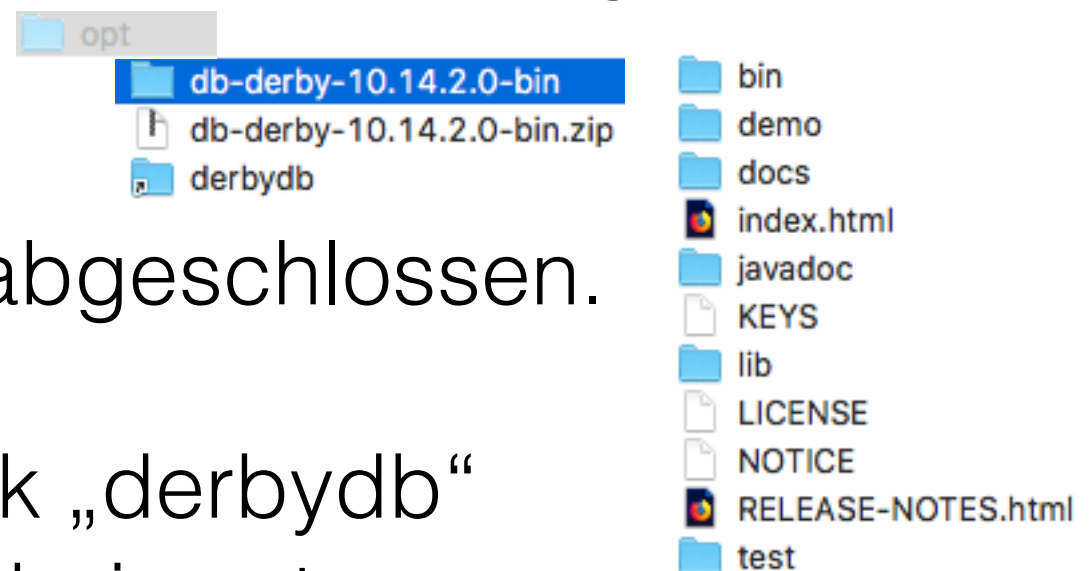
```
Terminal
+ Toms-MacBook-Pro:VehicleJdbc stuetz$ pwd
/Users/stuetz/Dropbox/htl/skripten/java.4jg.nvs/presentations/06.JPA/VehicleJdbc
- Toms-MacBook-Pro:VehicleJdbc stuetz$ mkdir db
Toms-MacBook-Pro:VehicleJdbc stuetz$ cd db
Toms-MacBook-Pro:db stuetz$ pwd
/Users/stuetz/Dropbox/htl/skripten/java.4jg.nvs/presentations/06.JPA/VehicleJdbc/db
Toms-MacBook-Pro:db stuetz$ $JAVA_HOME/db/bin/startNetworkServer -noSecurityManager
Sat Aug 29 09:06:48 CEST 2015 : Apache Derby Network Server - 10.11.1.2 - (1629631) st
rt 1527
```

- Erstellen eines db-Ordners im Projektverzeichnis
- Man wechselt in dieses Verzeichnis
- Starten der Datenbank (ohne SecurityManager)

ab Java 9 (jdk-9)

Datenbank downloaden

- Downloaden sie die binäre Distribution von https://db.apache.org/derby/derby_downloads.html zB db-derby-10.14.2.0-bin.zip
- Erstellen Sie im root Ihres Betriebssystems einen Folder „opt“ in diesen Ordner verschieben Sie obiges zip-File und entpacken dieses
- Die „Installation“ ist hiermit abgeschlossen.
- Ev. können Sie einen Symlink „derbydb“ auf das neu erstellte Verzeichnis setzen.



Datenbank starten

ab Java 9 (jdk-9)

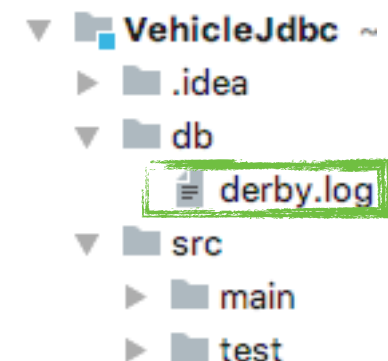
man öffnet in der IDE ein Terminal - somit befindet man sich im root-Verzeichnis des Projekts. Dort legt man ein Verzeichnis db an

In diesem Verzeichnis werden die DB-Files gespeichert.

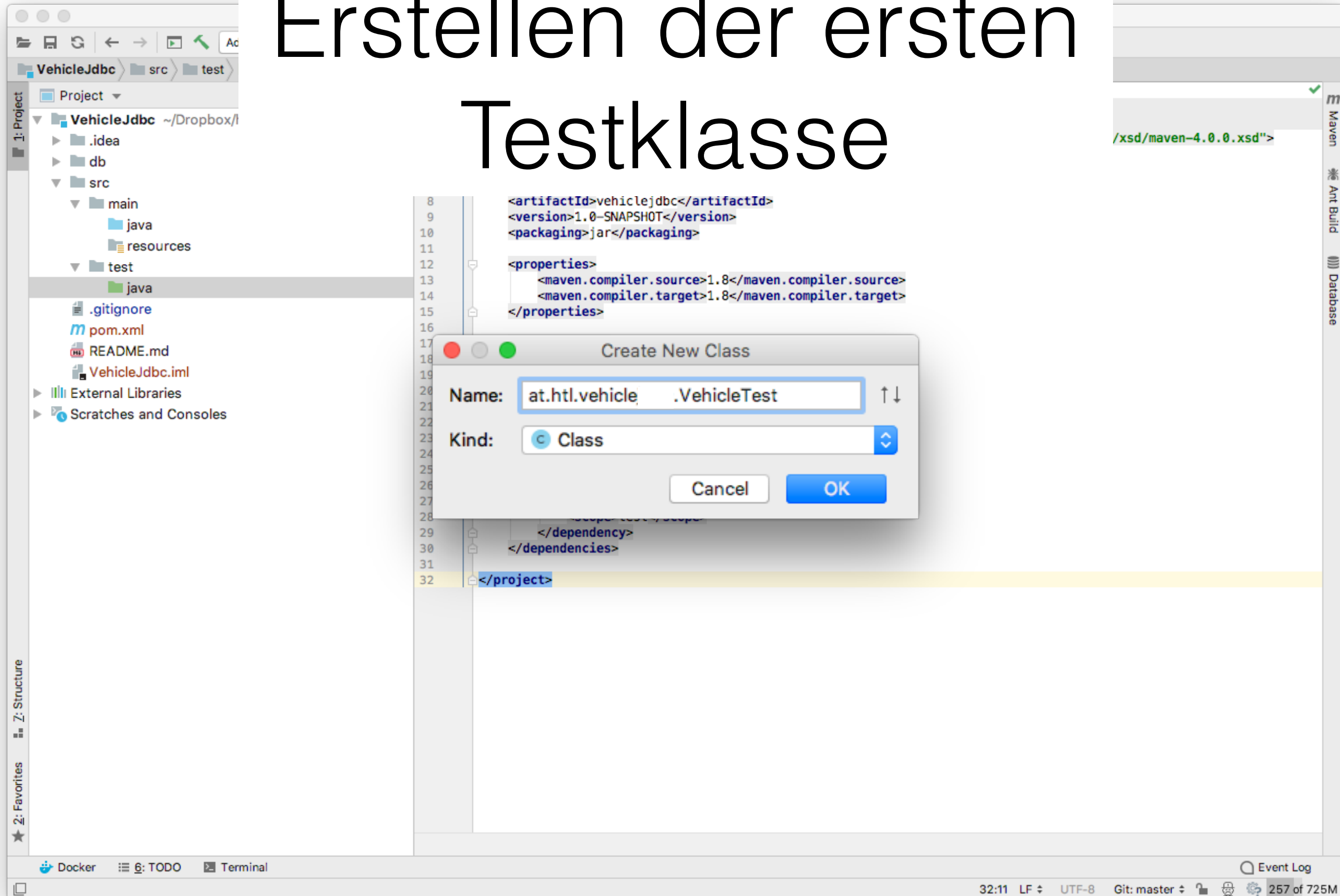
Terminal

```
+ Toms-MBP:VehicleJdbc stuetz$ java -version
x java version "10.0.1" 2018-04-17
Java(TM) SE Runtime Environment 18.3 (build 10.0.1+10)
Java HotSpot(TM) 64-Bit Server VM 18.3 (build 10.0.1+10, mixed mode)
Toms-MBP:VehicleJdbc stuetz$ pwd
/Users/stuetz/Dropbox/htl/skripten/themen/jdbc/projects/VehicleJdbc
Toms-MBP:VehicleJdbc stuetz$ mkdir db
Toms-MBP:VehicleJdbc stuetz$ cd db
Toms-MBP:db stuetz$ pwd
/Users/stuetz/Dropbox/htl/skripten/themen/jdbc/projects/VehicleJdbc/db
Toms-MBP:db stuetz$ /opt/derbydb/bin/startNetworkServer -noSecurityManager
Sun Oct 07 16:32:53 CEST 2018 : Apache Derby Network Server - 10.14.2.0 - (1828579) started and ready to accept connections on port 1527
```

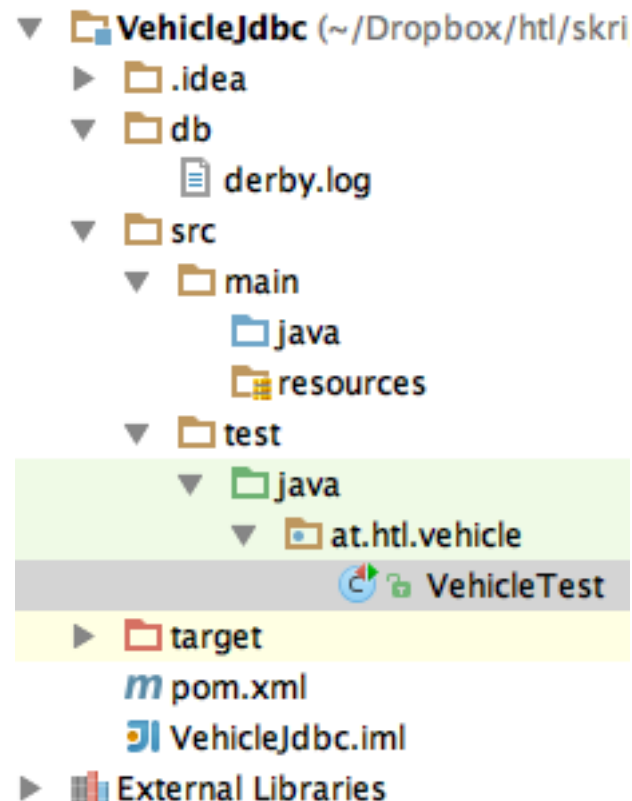
- „java -version“ gibt die aktuelle Java-Version aus
- „pwd“ gibt das aktuelle Verzeichnis aus
- die Datenbank wird ohne SecurityManager gestartet
- Kontrollieren Sie, ob die DB auch wirklich bereit ist und ob im db-Verzeichnis ein Log-File erstellt wurde



Erstellen der ersten Testklasse



Erstellen eines ersten Tests



```
package at.htl.vehicle;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import static junit.framework.TestCase.fail;

public class VehicleTest {

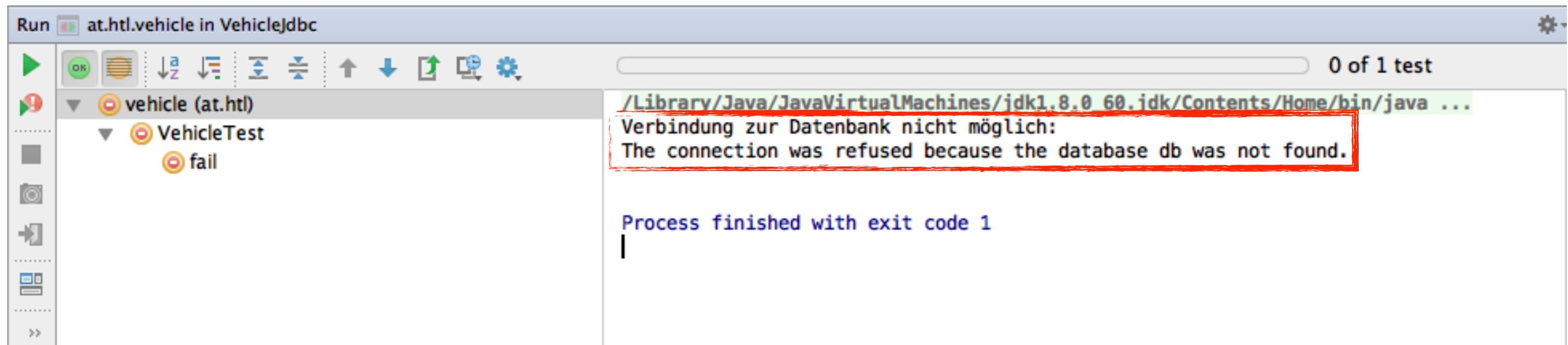
    public static final String DRIVER_STRING = "org.apache.derby.jdbc.ClientDriver";
    static final String CONNECTION_STRING = "jdbc:derby://localhost:1527/db";
    static final String USER = "app";
    static final String PASSWORD = "app";
    private static Connection conn;
```

```
@AfterClass
public static void teardownJdbc() {
    try {
        if(conn != null || !conn.isClosed()) {
            conn.close();
            System.out.println("Goodbye!");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

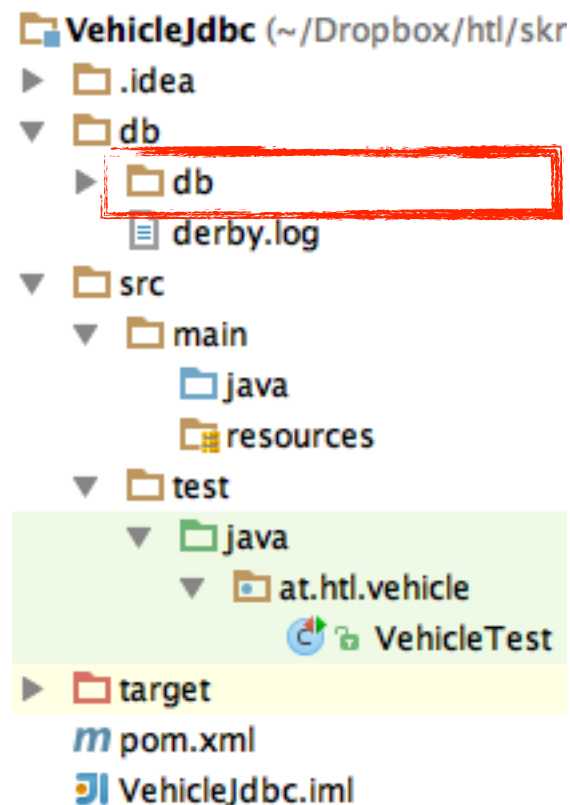
```
@BeforeClass
public static void initJdbc() {
    try {
        Class.forName(DRIVER_STRING);
        conn = DriverManager.getConnection(CONNECTION_STRING, USER, PASSWORD);
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        System.out.println("Verbindung zur Datenbank nicht möglich:\n"
            + e.getMessage() + "\n");
        System.exit(1);
    }
}
```

```
@Test
public void firstTest() {
    fail();
}
```

Testergebnis



```
static final String CONNECTION_STRING = "jdbc:derby://localhost:1527/db;create=true";
```



Nach neuerlichem
Starten des Tests
schlägt dieser immer
noch fehl, jedoch wird
eine Datenbank
angelegt

JDBC - DDL

```
@Test
public void ddl() {
    try {
        Statement stmt = conn.createStatement();

        String sql = "CREATE TABLE vehicle (" +
            "    id INT CONSTRAINT vehicle_pk PRIMARY KEY," +
            "    brand VARCHAR(255) NOT NULL," +
            "    type VARCHAR(255) NOT NULL)";

        stmt.execute(sql);
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}
```

Anmerkung: Dieser Test wird nie fehlschlagen, da kein Assert vorhanden ist. Es wird also nichts überprüft.

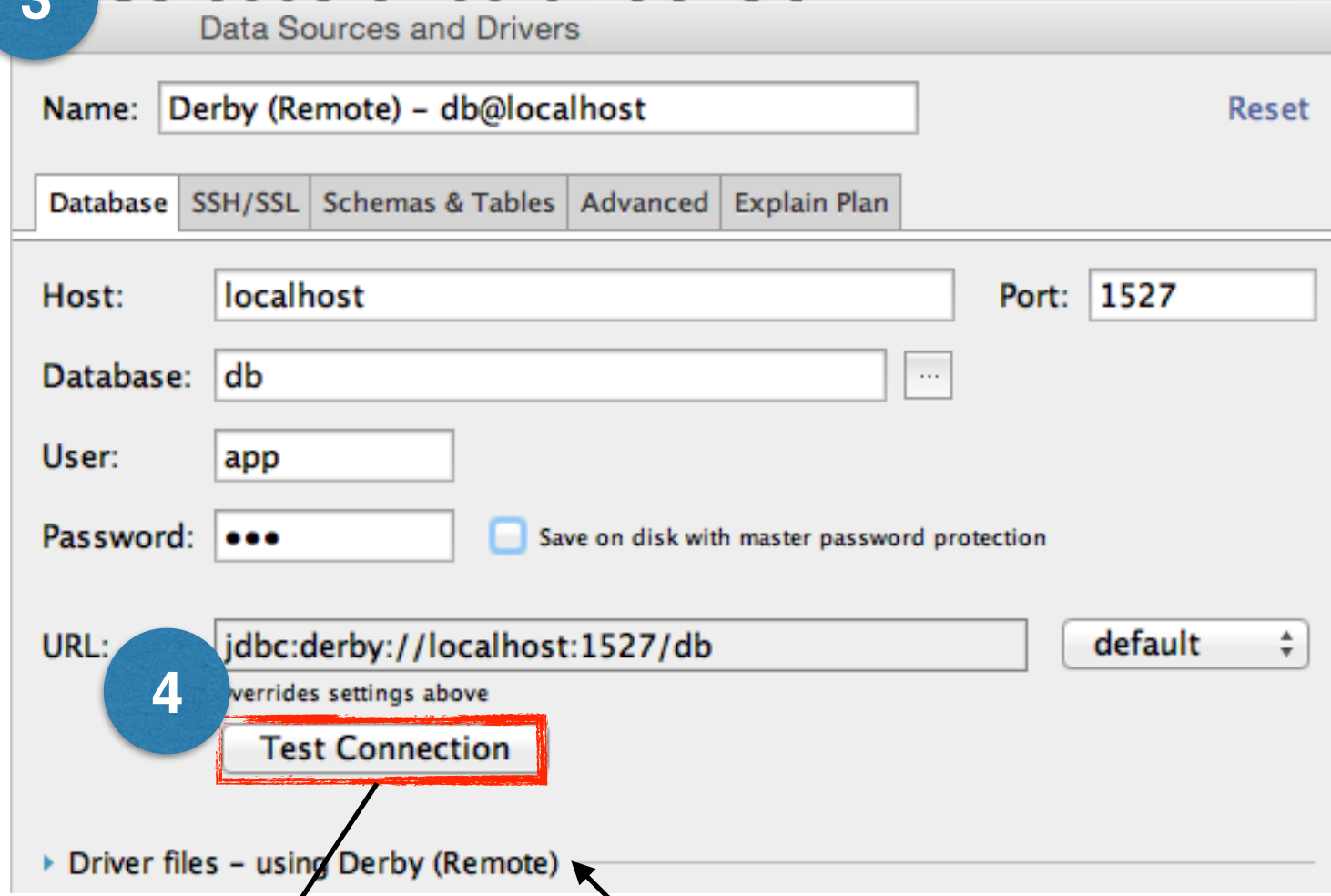
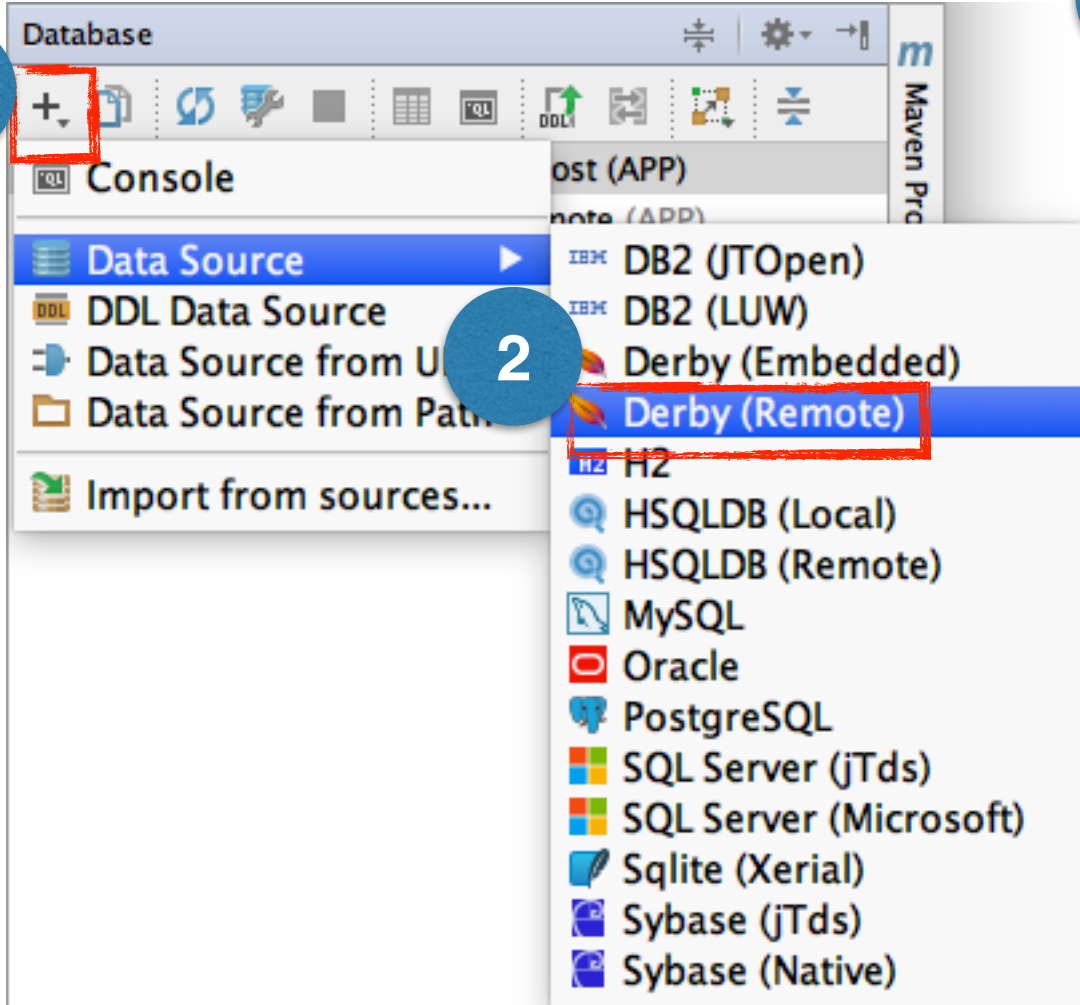
Datenquelle in IDE einrichten

1

2

3

4



Derby (Remote) - db@localhost

Host: localhost Port: 1527

Database: db

User: app

Password: ...

URL: jdbc:derby://localhost:1527/db

Test Connection

Driver files - using Derby (Remote)

Derby (Remote) - db@localhost

Database: Apache Derby/10.11.1.2 - (1629631)

Driver name: Apache Derby Network Client JDBC Driver

Driver version: 10.11.1.1 - (1616546)

JDBC version: 4.2

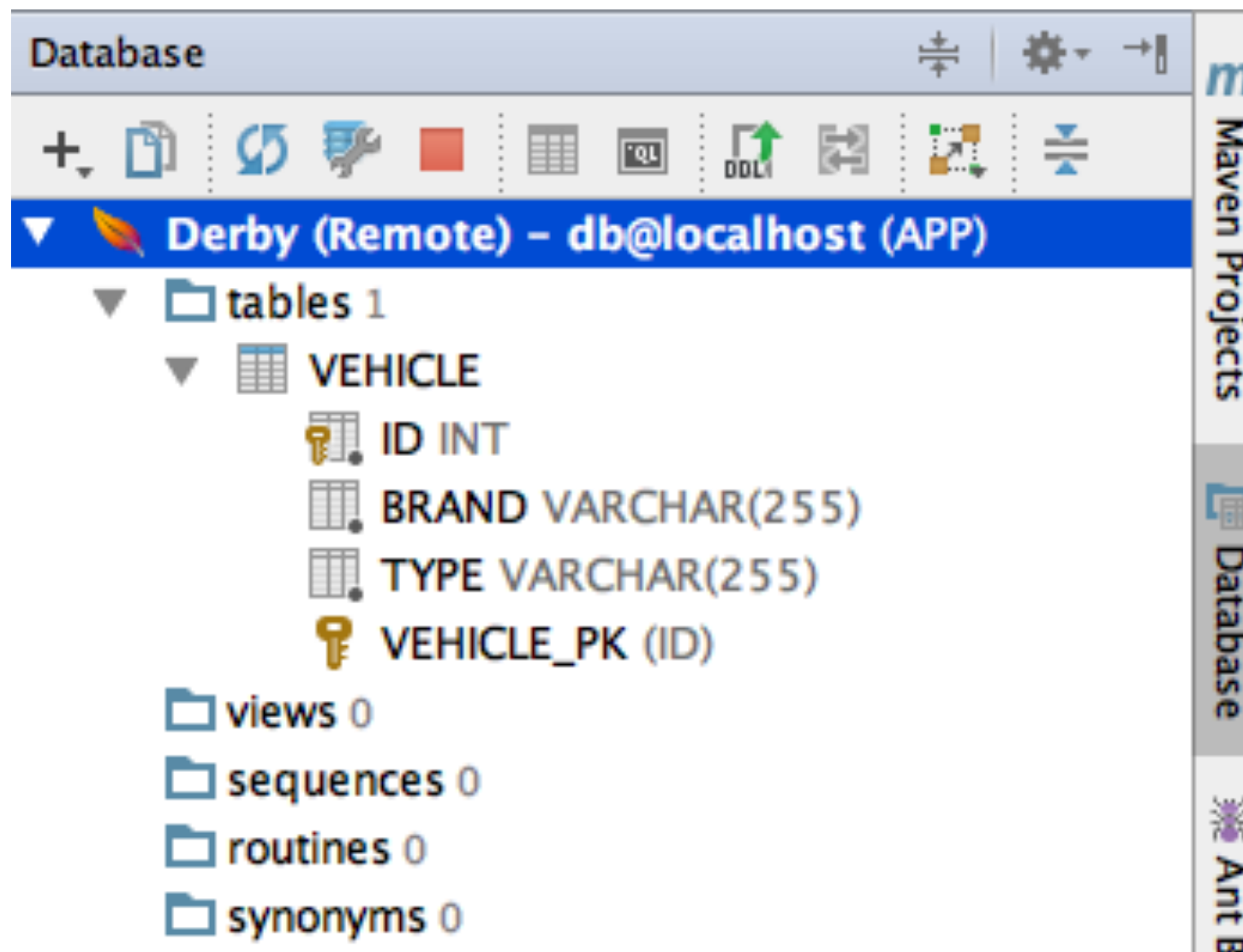
Case sensitivity: UPPER (quoted: EXACT)

Connection successful

OK

Eventuell muss man den Derby-Treiber downloaden, indem man den Link anklickt

Datasource



Testmethode

```
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;

@Test
public void dml() {

    // Daten einfügen
    int countInserts = 0;
    try {
        Statement stmt = conn.createStatement();
        String sql = "INSERT INTO vehicle (id, brand, type) VALUES (1,'Opel','Commodore')";
        countInserts += stmt.executeUpdate(sql);
        sql = "INSERT INTO vehicle (id, brand, type) VALUES (2,'Opel','Kapitän')";
        countInserts += stmt.executeUpdate(sql);
        sql = "INSERT INTO vehicle (id, brand, type) VALUES (3,'Opel','Kadett')";
        countInserts += stmt.executeUpdate(sql);
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
    assertThat(countInserts, is(3));

    // Daten abfragen
    try {
        PreparedStatement pstmt = conn.prepareStatement("SELECT id, brand, type FROM vehicle");
        ResultSet rs = pstmt.executeQuery();

        rs.next();
        assertThat(rs.getString("BRAND"), is("Opel"));
        assertThat(rs.getString("TYPE"), is("Commodore"));
        rs.next();
        assertThat(rs.getString("BRAND"), is("Opel"));
        assertThat(rs.getString("TYPE"), is("Kapitän"));
        rs.next();
        assertThat(rs.getString("BRAND"), is("Opel"));
        assertThat(rs.getString("TYPE"), is("Kadett"));
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Inhalt der Tabelle

APP.VEHICLE x

Database

3 rows

Comma-S

View Query

<Filter criteria>

	ID	BRAND	TYPE
1	1	Opel	Commodore
2	2	Opel	Kapitän
3	3	Opel	Kadett

Derby (Remote) - db@localhost (APP)

- tables 1
 - VEHICLE
 - ID INT
 - BRAND VARCHAR(255)
 - TYPE VARCHAR(255)
 - VEHICLE_PK (ID)
 - views 0
 - sequences 0
 - routines 0
 - synonyms 0

Setup

```
@BeforeClass
public static void initJdbc() {
    // Verbindung zur DB herstellen
    try {
        Class.forName(DRIVER_STRING);
        conn = DriverManager.getConnection(CONNECTION_STRING, USER, PASSWORD);
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        System.out.println("Verbindung zur Datenbank nicht möglich:\n"
            + e.getMessage() + "\n");
        System.exit(1);
    }

    // Erstellen der Tabelle VEHICLE
    try {
        Statement stmt = conn.createStatement();

        String sql = "CREATE TABLE vehicle (" +
            "    id INT CONSTRAINT vehicle_pk PRIMARY KEY," +
            "    brand VARCHAR(255) NOT NULL," +
            "    type VARCHAR(255) NOT NULL)";

        stmt.execute(sql);
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
}
```

Man könnte auch einen Autowert verwenden

```
String sql2 = "CREATE TABLE vehicle (" +
    "    id INT CONSTRAINT vehicle_pk PRIMARY KEY" +
    "    GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1)," +
    "    brand VARCHAR(255) NOT NULL," +
    "    type VARCHAR(255) NOT NULL)";
```

TearDown

```
@AfterClass
public static void teardownJdbc() {

    // Tabelle VEHICLE löschen
    try {
        conn.createStatement().execute("DROP TABLE vehicle");
        System.out.println("Tabelle VEHICLE gelöscht");
    } catch (SQLException e) {
        System.out.println("Tabelle VEHICLE konnte nicht gelöscht werden:\n"
            + e.getMessage());
    }

    // Connection schließen
    try {
        if (conn != null || !conn.isClosed()) {
            conn.close();
            System.out.println("Goodbye!");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```