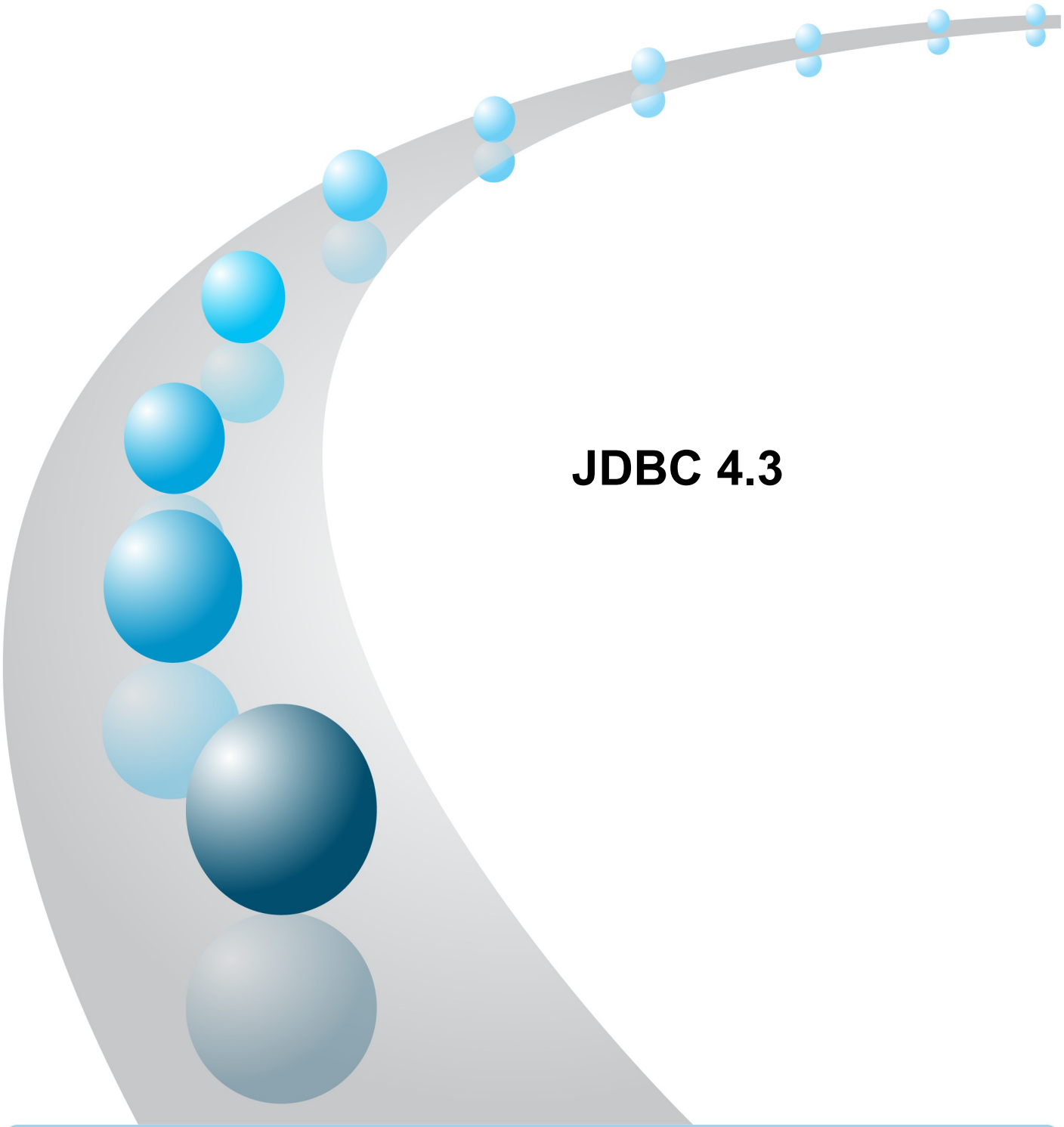
Noël Vaes

Java Trainer & Consultant



JDBC 4.3

Roode Roosstraat 5
3500 Hasselt
België

+32 474 38 23 94
noel@noelvaes.eu
www.noelvaes.eu

Correspondentie inzake overnemen of reproductie kunt u richten aan:

Noël Vaes
Roode Roosstraat 5
3500 Hasselt
België

Tel: +32 474 38 23 94

noel@noelvaes.eu
www.noelvaes.eu

24/12/2020

Inhoudsopgave

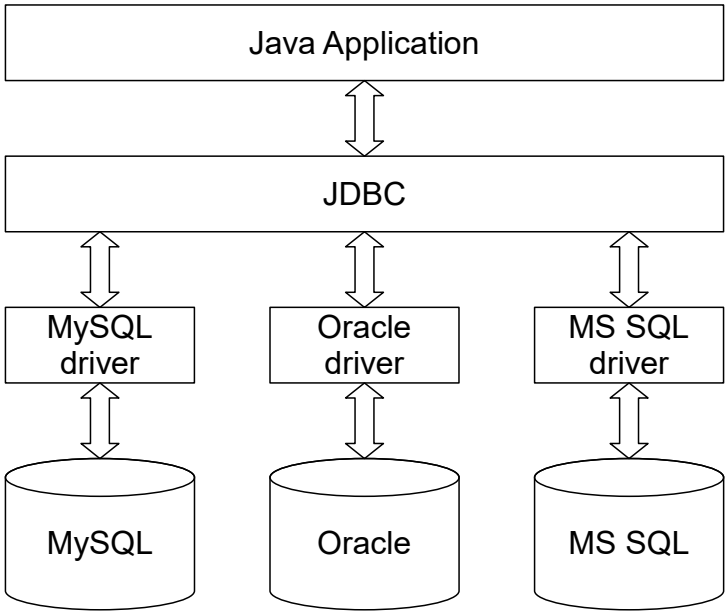
Hoofdstuk 1: JDBC.....4

1.1 Inleiding.....4
1.2 De databaseserver.....5
1.3 Een verbinding maken met een database.....5
1.3.1 De database-driver laden.....5
1.3.2 Een connectie maken.....7
1.4 SQL-commando's gebruiken.....10
1.4.1 Een statement creëren.....10
1.4.2 Gegevens uit een database opvragen 11
1.4.3 Gegevens wijzigen in een database.....16
1.4.4 Gegevens toevoegen aan de database.....17
1.4.5 Prepared statements.....18
1.4.6 De methode execute().....19
1.5 Transacties.....19
1.6 Batch updates.....23
1.7 Stored procedures.....23
1.8 Wijzigbare resultsets.....24
1.8.1 Een wijzigbare resultset creëren.....24
1.8.2 Gegevens in een rij wijzigen.....24
1.8.3 Rijen toevoegen en verwijderen.....25
1.9 Foutafhandeling.....26
1.9.1 SQLException.....26
1.9.2 SQLWarning.....26
1.10 Metadata.....27
1.10.1 ResultSetMetaData.....27
1.10.2 DatabaseMetaData.....27
1.11 Grote objecten.....27
1.11.1 Het lezen van grote objecten.....28
1.11.2 Het schrijven van grote objecten.....28
1.12 Data Access Objects (DAO).....28

Hoofdstuk 1: JDBC

1.1 Inleiding

JDBC is de afkorting van ***J**ava **D**ata**B**ase **C**onnectivity*. Het is een technologie die communicatie met allerlei relationele databases mogelijk maakt. JDBC biedt aan een Java-applicatie immers een uniforme interface naar databases van verschillende producenten. Om een bestaande database toegankelijk te maken via JDBC moet een specifieke *driver* geïnstalleerd worden. Voor de meest gangbare databases zijn er momenteel JDBC-*drivers* beschikbaar.



JDBC is vervat in het pakket **`java.sql`** uit de module **`java.sql`** die deel uitmaakt van de JSE. Meer geavanceerde mogelijkheden zijn ondergebracht in het pakket **`javax.sql`.**

Van JDBC zijn er verschillende versies die ondersteund worden in de volgende Java-versies:

JDK Versie	***JDBC Versie***
1.2	2.1
1.4	3.0
5	3.0
6	4.0
7	4.1
8	4.2
9	4.3

In de volgende paragrafen zullen we JDBC 4.3 behandelen.

1.2 De databaseserver

Databaseservers zijn er allerhande. Zo hebben we bekende producten als *Oracle* en *Microsoft SQL Server*. Voor huis-, tuin- en keukengebruik op *Microsoft*-systemen wordt ook wel *Access* gebruikt.

In de o*pen-source*-wereld wordt onder andere gebruikgemaakt van *MySQL.* Deze databaseserver kan vrij gedownload worden van *www.mysql.com*. Hiervan bestaat een binair compatibele variant *MariaDB* die te vinden is op *www.mariadb.org*. In het verdere verloop van deze cursus zullen we gebruikmaken van *MariaDB* maar alle voorbeelden zijn ook toepasbaar op andere databaseservers.

Er kan gebruikgemaakt worden van de volgende database die beschikbaar is op het internet:

- hostname**: noelvaes.eu**
- login**: student**
- wachtwoord**: student123**
- databank: **StudentDB**

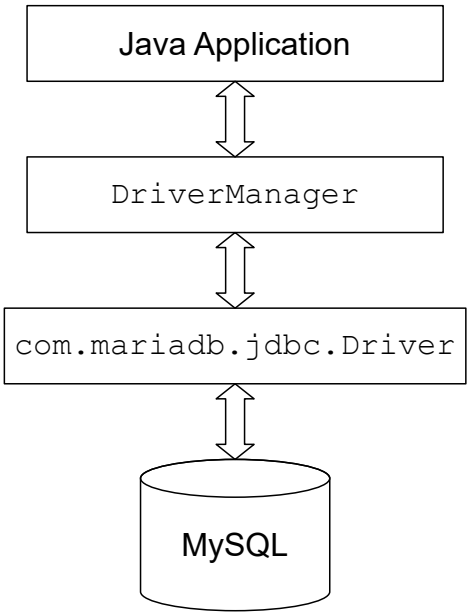
Om de tabellen te beheren of nieuwe aan te maken kan je eventueel gebruikmaken van ***MySQL Workbench.***

1.3 Een verbinding maken met een database

1.3.1 De database-*driver* laden

Aangezien JDBC een universele interface is naar om het even welke database, verloopt de communicatie via een laag die volledig abstractie maakt van de onderliggende database. Hierbij speelt de klasse `java.sql.DriverManager` een cruciale rol. Het opzetten van de communicatie met de database verloopt via deze klasse, die alleen maar statische methoden heeft.

Specifieke database-*drivers*, zoals die voor *MariaDB* moeten zich registreren bij de `DriverManager` klasse.



Deze registratie kan gebeuren met de volgende methode:

```
DriverManager.registerDriver().
```

Van JDBC-*drivers* wordt echter verondersteld dat ze zichzelf registreren bij de *DriverManager* zodra ze in de JVM worden ingeladen. Deze registratie hoeft de gebruiker van de *driver* dus niet zelf te doen. Men moet er alleen voor zorgen dat de *driver* ingeladen wordt in de JVM, de rest gebeurt dan vanzelf.

Om de *driver* in te laden wordt er gebruikgemaakt van de methode `Class.forName()` die een klassenobject van een bepaalde klasse maakt. Het neveneffect van deze methode is uiteraard dat de klasse in de JVM geladen wordt, indien dat tenminste nog niet gebeurd was. Dit neveneffect wordt meestal gebruikt om *JDBC-drivers* te laden.

Bij het inladen van een klasse worden de `static` codeblokken uitgevoerd en de klasse van een *JDBC-driver* zal in zo'n codeblok zichzelf registreren bij de `DriverManager`.

Een lijst of *stream* van geregistreerde *drivers* kan opgevraagd worden met respectievelijk `DriverManager.getDrivers()` en `DriverManager.drivers()`.

Voor de *MariaDB*-database is de *driver* een klasse met de naam **org.mariadb.jdbc.Driver**. Deze klasse vinden we in het JAR-bestand ***mariadb-java-client-x.y.z.jar*** dat gedownload kan worden van de *MariaDB*-site (https://downloads.mariadb.org/).

We kunnen deze driver dan ook als volgt inladen:

```
Class.forName("org.mariadb.jdbc.Driver");
```

De methode `Class.forName()` gooit een `ClassNotFoundException` indien de opgegeven klasse niet gevonden wordt in het ***module path***. We moeten het JAR-bestand met de *driver* daarom ook opnemen in ons *module path*.

Elke database heeft zijn eigen *driver*-klasse die op deze wijze geladen moet worden. Voor de precieze naam van de *driver*-klasse moet men de bijgeleverde documentatie raadplegen.

Sinds JDK 6 en JDBC 4.0 is dit laden van de driver evenwel niet meer expliciet nodig. Men komt het wel nog vaak tegen in oude code die gemaakt is voor JDK 6. In sommige omstandigheden werkt dit automatisch laden evenwel niet en moeten we toch terugvallen op de klassieke techniek. Dit is onder andere het geval indien we een webapplicatie maken.

1.3.2 Een connectie maken

Zodra de database-*driver* geladen is, kunnen we een connectie maken met de database. Deze connectie wordt nadien gebruikt om te communiceren met de database.

Een connectie wordt gemaakt met de volgende methode van de klasse `DriverManager`

```
DriverManager.getConnection(url,login,password)
```

De methode neemt als eerste parameter een string met de database-URL. Deze URL heeft de volgende vorm:

```
jdbc:subprotocol:subname
```

Het *subprotocol* geeft aan welke *driver* de *DriverManager* moet gebruiken. Iedere geregistreerde *driver* heeft zijn eigen subprotocol. De *subname* is extra informatie die

doorgegeven wordt aan de betreffende *driver*. Meestal bevat deze de naam van de *host* en de naam van de database.

De precieze URL hangt dus af van de database waarmee men communiceert. Voor de database die we gecreëerd hebben is dat het volgende protocol:

```
jdbc:mariadb://noelvaes.eu/StudentDB
```

De tweede parameter van de methode `getConnection()` is de gebruikersnaam en de derde parameter is het wachtwoord van de gebruiker.

De methode `getConnection()` geeft een object terug van de interface `Connection`. Dit object representeert de geopende connectie met de database.

We kunnen dus als volgt een connectie maken met de *MariaDB* database:

```
Connection con = DriverManager.getConnection
("jdbc:mariadb://noelvaes.eu/StudentDB","student","student123");
```

De interface `Connection` is afgeleid van `AutoCloseable` zodat we hier gebruik kunnen maken van een *try with resources*. De connectie wordt dus automatisch afgesloten na het *try*-blok.

De code voor ons eerste programma ziet er als volgt uit.

```
package eu.noelvaes.jdbc;

import java.sql.*;
public class ConnectDB {
   public static void main(String[] args) {
      try (Connection con = DriverManager.getConnection(
            "jdbc:mariadb://noelvaes.eu/StudentDB","student",
            "student123")) {
         System.out.println("Connection OK");
      }
      catch (Exception ex) {
         System.out.println("Oops, something went wrong!");
         ex.printStackTrace(System.err);
      }
   }
}
```

Het laden van de *driver* laten we hier achterwege omdat dit sinds JDK 1.6 niet meer nodig is.

We maken onmiddellijk een connectie. Deze methode kan een *exception* genereren zodat we de nodige afhandeling daarvoor moeten voorzien.

Verder dienen we het pakket `java.sql` te importeren. Dat pakket hoort bij de module `java.sql`. We voegen aan ons project daarom de volgende modulebeschrijving toe:

module-info.java

```
module eu.noelvaes.jdbc {
    requires java.sql;
}
```

Om het programma uit te voeren moeten we tevens de pakketten van *MariaDB* in ons

module path opnemen.

```
java -p mariadb-java-client-x.y.z.jar;. ^
     -m eu.noelvaes.jdbc/eu.noelvaes.jdbc.ConnectDB
```

Opdracht 1: Een connectie maken met de database

In deze opdracht maken we een programma dat de JDBC-*driver* laadt en een connectie maakt met de database.

- Maak een nieuw Java-project in je IDE.
- Voeg het JAR-bestand van de *MariaDB-driver* aan je project toe. Indien je gebruikt maakt van *Maven* kan je de volgende *dependency* toevoegen aan de POM:

```
<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>eu.noelvaes.jdbc</groupId>
   <artifactId>JDBC</artifactId>
   <version>1.0</version>
   <name>JDBC Solution</name>
   <properties>
      <maven.compiler.source>11</maven.compiler.source>
      <maven.compiler.target>11</maven.compiler.target>
      <project.build.sourceEncoding>
         UTF-8
      </project.build.sourceEncoding>
   </properties>
   <build>
      <finalName>${project.artifactId}</finalName>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
         </plugin>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>3.0.0-M5</version>
         </plugin>
      </plugins>
   </build>
   <dependencies>
      <dependency>
         <groupId>org.mariadb.jdbc</groupId>
         <artifactId>mariadb-java-client</artifactId>
         <version>2.7.0</version>
      </dependency>
      <dependency>
         <groupId>org.junit.jupiter</groupId>
         <artifactId>junit-jupiter-engine</artifactId>
         <version>5.7.0</version>
         <scope>test</scope>
```

```
      </dependency>
   </dependencies>
</project>
```

- Open de documentatie van het pakket `java.sql` en zoek de beschrijving van de klasse `DriverManager` en de interface `Connection`.
- Voeg de modulebeschrijving ***module-info.java*** aan je project toe:

```
module eu.noelvaes.jdbc {
    requires java.sql;
}
```

- Maak een nieuw bestand met de naam ***ConnectDB.java*** en geef de volgende code in:

```
package eu.noelvaes.jdbc;

import java.sql.*;
public class ConnectDB {
   public static void main(String[] args) {
      try (Connection con = DriverManager.getConnection(
            "jdbc:mariadb://noelvaes.eu/StudentDB","student",
            "student123")) {
         System.out.println("Connection OK");
      }
      catch (Exception ex) {
         System.out.println("Oops, something went wrong!");
         ex.printStackTrace(System.err);
      }
   }
}
```

- Compileer het programma en voer het uit.
- Optioneel: maak een unit-test waarin de connectie wordt aangemaakt in de `@BeforeAll` en gesloten in de `@AfterAll`.

1.4 SQL-commando's gebruiken

1.4.1 Een statement creëren

SQL (***S**tructured **Q**uery **L**anguage*) is de meest gebruikte taal om te communiceren met een database. JDBC biedt de mogelijkheid SQL-commando's naar de database te sturen. Hiervoor gebruiken we een object van de interface `Statement` dat een SQL-commando bevat. Zo'n object wordt gemaakt met de methode `createStatement()` van de interface `Connection`.

Concreet ziet dit er als volgt uit:

```
try (Connection con = DriverManager.getConnection(

"jdbc:mariadb://noelvaes.eu/StudentDB","student","student123");
     Statement stmt = con.createStatement()) {
   ...
```

```
}
```

Het statement dient na gebruik afgesloten te worden met de methode `close()`. Aangezien de interface `AutoCloseable` geïmplementeerd wordt, kunnen we ook hier gebruikmaken van de *try with resources*.

Het *statement*-object kan op verschillende manieren SQL-commando's naar de database sturen. In de volgende tabel worden enkele methoden van de interface `Statement` opgesomd. Voor een volledige en precieze omschrijving verwijzen we naar de Java-documentatie.

Methode	***Omschrijving***
`executeUpdate(String sql)`	Voert een CREATE-, INSERT-, UPDATE- of DELETE-commando uit.
`executeQuery(String sql)`	Voert een commando uit dat als resultaat een tabel heeft (*resultset)*. Dit is typisch het geval bij een SELECT-commando.
`executeBatch()`	Voert een reeks van UPDATE-commando's uit. De commando's worden toegevoegd met de methode `addBatch()`.
`execute()`	Voert om het even welk SQL-commando uit.

De methoden `executeUpdate()` en `executeQuery()` hebben als parameter een tekenreeks die het SQL-commando bevat.

De methode `executeUpdate()` geeft een integer als resultaat terug die het aantal rijen bevat waarin een aanpassing is gebeurd.

De methode `executeBatch()` geeft een reeks van integers terug, waarbij elk element het resultaat is van de overeenkomstige opdracht in de *batch.*

De methode `executeQuery()` geeft als resultaat een `ResultSet` terug die de gegevens bevat van de zoekopdracht.

De methode `execute()` geeft een `boolean` terug die aangeeft of het resultaat al dan niet een `ResultSet` is.

In het verdere verloop van de cursus komen deze verschillende methoden één voor één aan bod.

1.4.2 Gegevens uit een database opvragen

Gegevens worden uit een database opgevraagd met het SQL-commando SELECT:
Om alle gegevens op te vragen uit de tabel ***Beers*** gebruiken we het volgende commando:

```
SELECT * FROM Beers
```

Om het commando uit te voeren, gebruiken we de methode `executeQuery()`. Deze methode geeft een object van de interface `ResultSet` terug.

```
String sql = "SELECT * FROM Beers";
try (Connection con = DriverManager.getConnection(
    "jdbc:mariadb://noelvaes.eu/StudentDB","student","student123");
```