Persistentie

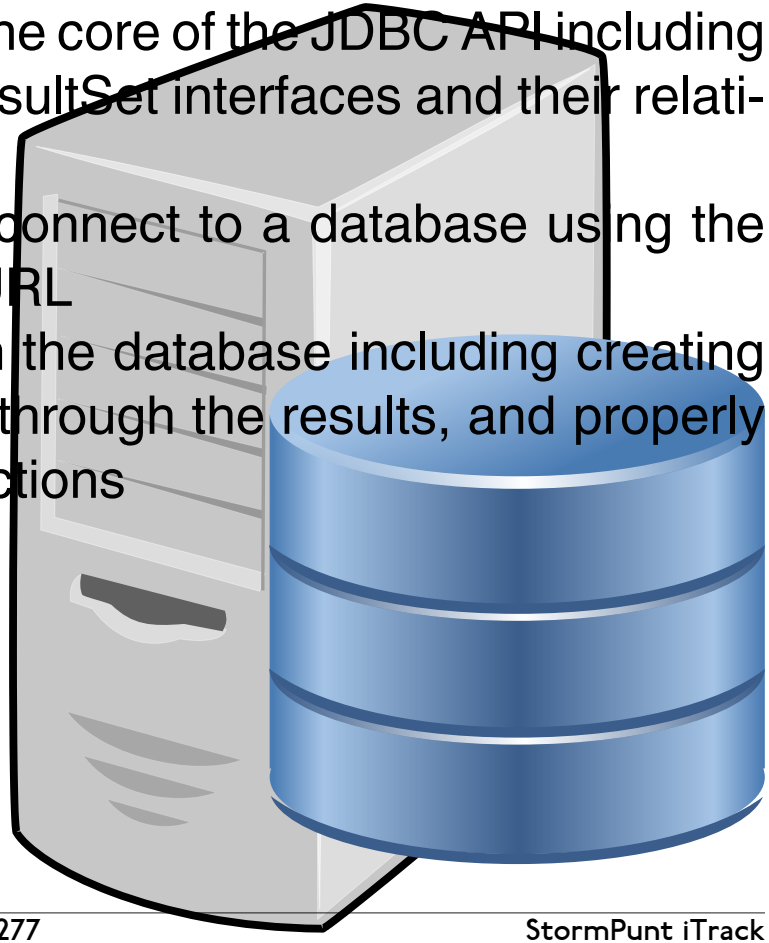
11

Leerdoelen:

11.1 Describe the interfaces that make up the core of the JDBC API including the Driver, Connection, Statement, and ResultSet interfaces and their relationship to provider implementations

11.2 Identify the components required to connect to a database using the DriverManager class including the JDBC URL

11.3 Submit queries and read results from the database including creating statements, returning result sets, iterating through the results, and properly closing result sets, statements, and connections



Facade

Zoals in de introductie is aangegeven zijn er heel veel mogelijkheden om persistentie te realiseren. Als een programma persistentie nodig heeft zal dat programma niet geïnteresseerd zijn in de compliciteit van het persistentie mechanisme. Het is dan goed om die compliciteit af te schermen en gebruik te maken van het "Facade"-pattern.

Het design pattern "Facade" behoort tot de categorie "Structural Design Patterns". *Het design pattern 'Facade' wordt beschreven in slide Paragraaf 5.5.4, “Design Pattern: Facade” en verder.*

Data Access Object

Er is een verbijzondering in Java Enterprise architectuur gemaakt van het "Facade"-pattern, namelijk het "Data Access Object"-pattern. Voor relationele databases is een speciale specificatie, deze is al genoemd in de introductie, namelijk JPA. Hierdoor is het standaard geïmplementeerd in Java Enterprise Edition 5 door middel van de interface*javax.persistence.EntityManager* .JPA wordt binnen deze leereenheid niet gebruikt.

Het design pattern "Data Access Object" behoort tot de categorie "Integration Tier".

Connection URL's

```
jdbc:odbc:northwind
```

2

- jdbc; is het protocol en is verplicht voor alle jdbc URL's.
- odbc; is hier het subprotocol en geeft in dit geval aan dat de JDBC – ODBC bridge gebruikt moet worden. In het algemeen geeft het subprotocol aan welk database systeem gebruikt wordt (DBMS).
- northwind; geeft aan welke database er gebruikt gaat worden. In het geval er gebruik wordt gemaakt van een database server, wordt daar vaak de naam van deze machine en een eventueel poortnummer aan toegevoegd. Voor een MySQL database zou de URL er als volgt uit kunnen zien:

```
jdbc:mysql://localhost:3306/elpo
```

2

java.sql.DriverManager

De DriverManager class kent alle JDBC-drivers in de class loader, en functioneert daarmee als een SPI.

JDBC Driver properties:

```
jdbc.drivers=COM.cloudscape.core.JDBCDriver:com.oracle.jdbc.OracleDriver.
2
```

JDBC Driver properties via code:

```
System.setProperty(
    "jdbc.drivers",
    "COM.cloudscape.core.JDBCDriver:com.oracle.OracleDriver");

```

Voorbeeld 1; connectie maken:

```
try {
    Class.forName("Driver class name");
    Connection connection = DriverManager.getConnenction("jdbcurl");
    // hier kun je verder met de connection
} catch(SQLException e) {
 // Loggen !!!!
}

```

Voorbeeld 2; connectie maken:

```
try {
    Class.forName("Driver class name");
    Connection connection = DriverManager.getConnenction("jdbcurl","username","p
    // hier kun je verder met de connection
} catch(SQLException e) {
 // Loggen !!!!
}

```

Voorbeeld 3; connectie maken:

```
try {
    Class.forName("Driver class name");
    Properties properties = null; //properties ergens vandaan halen
    Connection connection = DriverManager.getConnenction("jdbcurl",properties);
    // hier kun je verder met de connection
} catch(SQLException e) {
 // Loggen !!!!
}
```

Voorbeeld 4; connectie met ARM:

```
Class.forName("Driver class name");
Properties properties = null; //properties ergens vandaan halen

try (  Connection connection =
            DriverManager.getConnenction("jdbcurl",properties);
     ) {

  // hier kun je verder met de connection

} catch(SQLException e) {
 // Loggen !!!!
}
```

Overzicht Statement interfaces

- **java.sql.Statement.** Een basis SQL statement.
- **java.sql.PreparedStatement.** Een voorgecompileerd SQL statement. Gebruik hiervan kan de verwerkingssnelheid ten goede komen
- **java.sql.CallableStatement.** Toegang tot "Stored procedures" van de database.

Statement voorbeeld:

```
Statement queryStat = conn.createStatement();
ResultSet rs = queryStat.executeQuery("SELECT * FROM werknemers");
Statement updateStat = conn.createStatement();
int rijen = updateStat.executeUpdate("UPDATE werknemers SET \
                    dienstjaren = dienstjaren + 1 WHERE id = 12");

```

PreparedStatement voorbeeld:

```
PreparedStatement updateStat = conn.prepareStatement("UPDATE \
      werknemers SET dienstjaren = dienstjaren + 1 WHERE id = ?");
updateStat.clearParameters(); //nu niet nodig ivm nieuwe instance.
updateStat.setString(1,”12”);
updateStat.executeUpdate();

```

CallableStatement voorbeeld:

```
CallableStatement callStat = conn.prepareCall("{call procedure-naam(?,?)}");
2
```

Het aanroepen van een stored procedure:

```
CallableStatement callStat = connection.prepareCall({
                             "call updateWerknemer(?)}");
callStat.registerOutParameter(2, Types.INTEGER);
callStat.setInt(1,12);
callStat.execute();
System.out.print("Rows affected: " + callStat.getInt(2);
```

javax.sql.RowSet

De RowSet interface is een verbijzondering van de ResultSet interface. Om het leven van de developer te ondersteunen is de RowSet meer toegeschreven naar de JavaBean en zijn er extra functionaliteit gemaakt voor het scrollen en het updaten in de ResultSet zelf.