ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

JDBC
Java Database Connectivity

Αθ. Ανδρούτσος

JDBC (1)

- Το **Java Database Connectivity (JDBC)** είναι ένα standard API (JDBC API) που περιλαμβάνει προδιαγραφές (interfaces) για την επικοινωνία εφαρμογών Java με Βάσεις Δεδομένων. Δύο packages, **java.sql, javax.sql**
- Από τη στιγμή που το JDBC είναι specification χρειάζεται και μία υλοποίηση από το πρόγραμμα που θέλει να το χρησιμοποιήσει, χρειάζεται δηλαδή ένας Driver για τη συγκεκριμένη Database που θέλουμε να συνδεθούμε

JDBC (2)

- Ο Driver είναι ένα πρόγραμμα που κάνει τη μετάφραση μεταξύ μίας ΒΔ και της Java (π.χ. **MySQL J/Connector**)
- Ένα πρόγραμμα Java που θέλει να χρησιμοποιήσει το JDBC και να συνδεθεί με μία ΒΔ πρέπει να κάνει Load τον συγκεκριμένο Driver (π.χ. MySQL J/Connector) πριν συνδεθεί στην DB

JDBC API

- Η τρέχουσα έκδοση του JDBC είναι η JDBC 4.3
- Το JDBC API ορίζεται στο package java.sql και υλοποιείται από τους αντίστοιχους connectors/drivers για κάθε τεχνολογία ΒΔ
- Για παράδειγμα ο **DriverManager** είναι μία κλάση στο **java.sql** package που δίνει μία μέθοδο, την **DriverManager.getConnection()** για σύνδεση σε μία ΒΔ με ένα URL και με credentials

Driver Register

- Στο JDBC 4.0 στο Java SE ο driver φορτώνεται αυτόματα. Σε κάθε περίπτωση μη αυτόματη φόρτωση του Driver μπορεί να γίνει με κλήση στην **Class.forName()** αφού πρώτα έχουμε κατεβάσει στα Dependencies το jar που τον περιέχει

```
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    conn = DriverManager.getConnection(url,  username, password);
    System.out.println("Connection Established");
} catch (SQLException e1) {
    e1.printStackTrace();
} catch (ClassNotFoundException e1) {
    e1.printStackTrace();
}
```

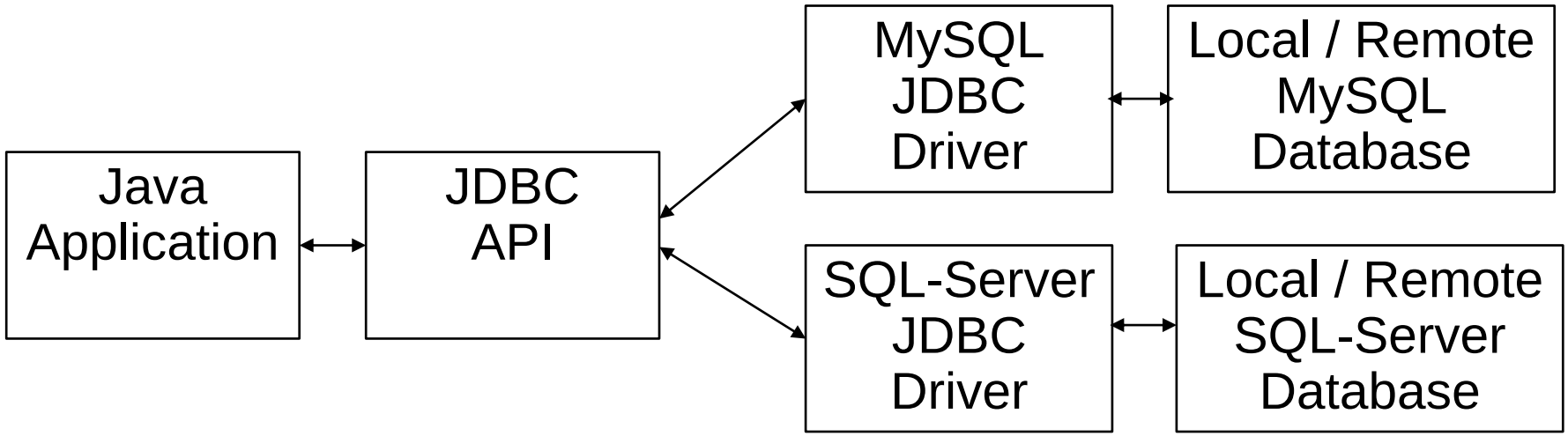
JDBC Driver (1)

- Υλοποιεί την επικοινωνία Java - MySQL
- Ο **MySQL Connector/J** είναι ο επίσημος JDBC driver για MySQL
- Η τελευταία έκδοση 9.0.xx κατεβαίνει μαζί με το MySQL Server ή ανεξάρτητα από: https://dev.mysql.com/downloads/connector/j/
- Παλαιότερες εκδόσεις μπορούν να κατέβουν από: https://downloads.mysql.com/archives/c-j/

JDBC Driver (2)

- Υλοποιεί τις SQL εντολές πάνω από Java Sockets, χρησιμοποιώντας την σουίτα πρωτοκόλλων TCP/IP για την επικοινωνία με τον MySQL Server
- Η επικοινωνία γίνεται στη θύρα *TCP/3306* (default port)
- Πρόκειται για Type 4 Driver: native-Protocol Driver για συγκεκριμένο Data Source (https://www.ibm.com/docs/en/i/7.4?topic=jdbc-types-drivers)

Εφαρμογές με JDBC



- Για επικοινωνία με κάθε διαφορετική ΒΔ χρειαζόμαστε διαφορετικό driver

CRUD

- Πρόκειται για ακρωνύμιο που αντιστοιχεί στις τέσσερις βασικές πράξεις που μπορούμε να κάνουμε σε μία ΒΔ
  - Create
  - Read
  - Update
  - Delete

CRUD/SQL Αντιστοίχιση

CRUD	SQL
Create	**INSERT** INTO Teachers (ID, S_NAME, F_NAME) VALUES(1, '…', '…');
Read	**SELECT** * FROM Teachers;
Update	**UPDATE** Teachers SET S_NAME = 'PAP' WHERE ID=1;
Delete	**DELETE** FROM Teachers WHERE ID=1;

- Έστω ο πίνακας TEACHERS με πεδία
  - ID INT
  - S_NAME VARCHAR(45)
  - F_NAME VARCHAR(45)

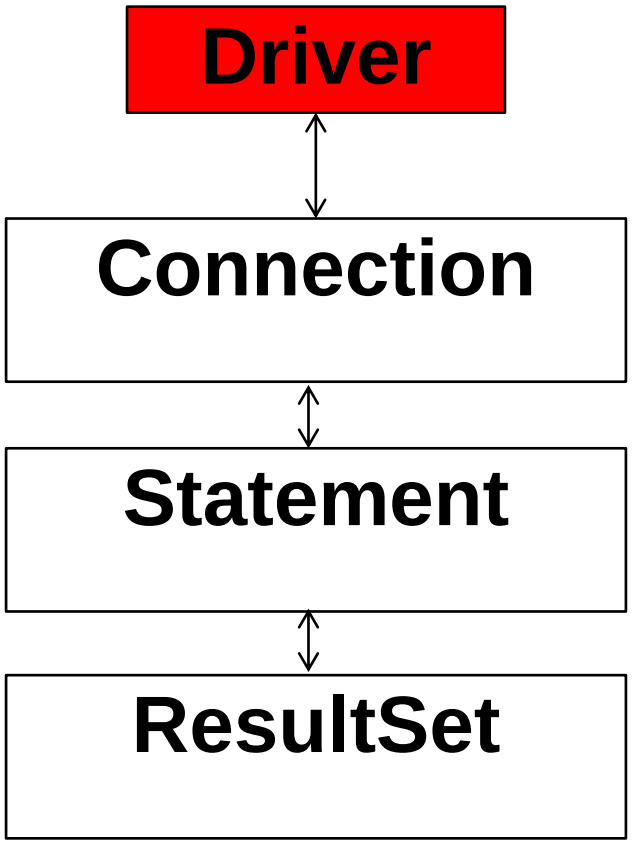
Workflow

1. Σύνδεση με τη ΒΔ (Connection)
2. SQL Εντολές CRUD (Statements)
3. Λήψη των αποτελεσμάτων του QUERY και επεξεργασία (Result Set

Workflow JDBC (1)

1. import packages
2. Δημιουργία Connection
3. Δημιουργία Statement
4. Εκτέλεση Statement
5. Read και process ResultSet object
6. Close ResultSet, Statement, Connection

Workflow JDBC (2)



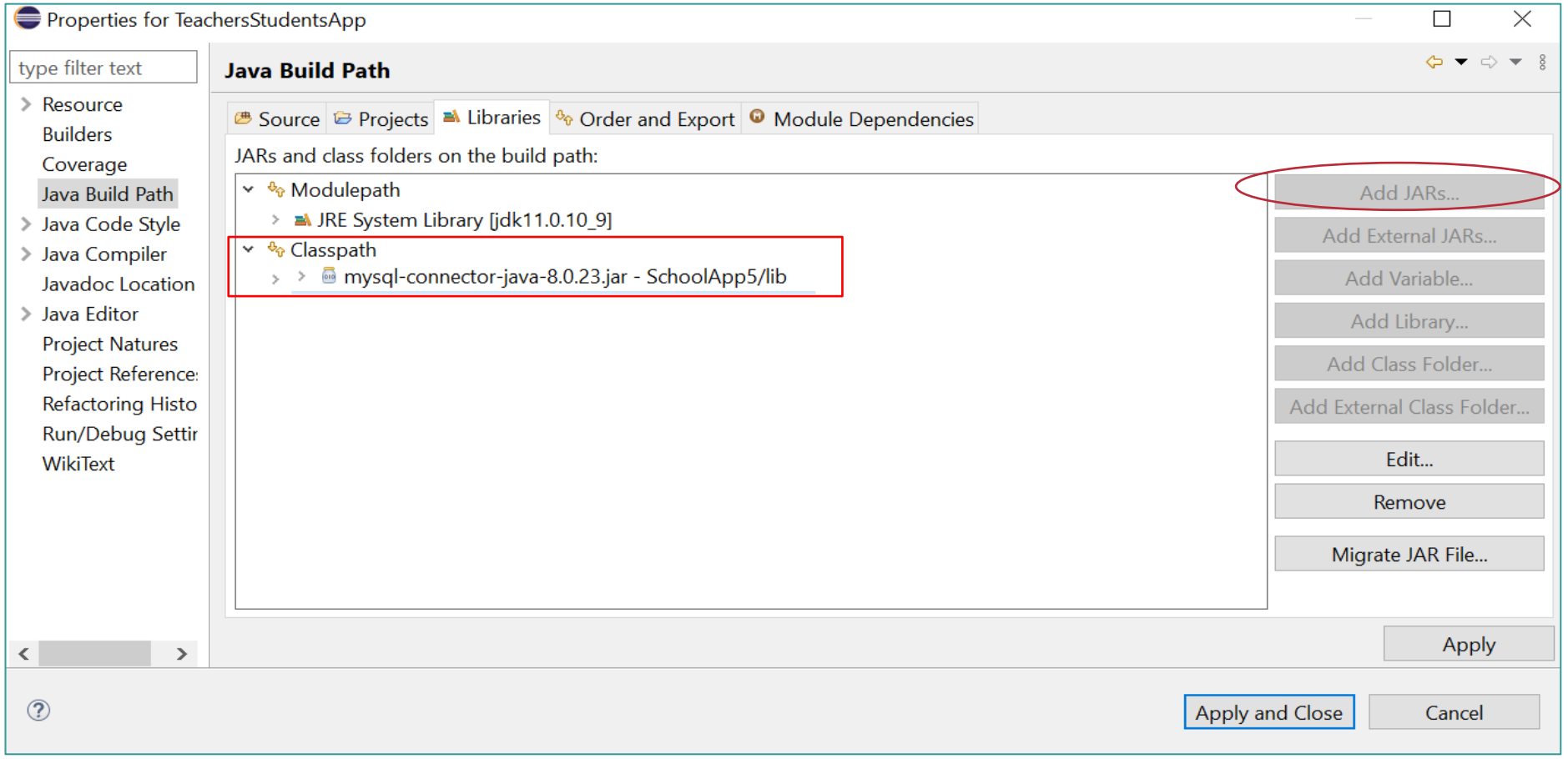
DriverManager

- Για να συνδεθεί μία εφαρμογή Java σε μία ΒΔ χρησιμοποιούμε την κλάση:
  - ***DriverManager***, που συνδέει μία εφαρμογή με ένα Data Source, που προσδιορίζεται με ένα Database URL. Η σύνδεση χρειάζεται επίσης properties, όπως username και password
  - Για να συνδεθούμε με μία ΒΔ, χρησιμοποιούμε τη μέθοδο **DriverManager.getConnection() που επιστρέφει ένα Connection object.** Όταν ο DriverManager προσπαθεί να συνδεθεί στη ΒΔ, **φορτώνει αυτόματα τον JDBC Driver** που έχουμε εισάγει στο build-path του IDE (στην περίπτωση που αυτός είναι στο Buildpath/Classpath)
  - Διαφορετικά, αν ο Driver δεν βρίσκεται στο Buildpath (όπως όταν συνδεόμαστε μέσω Apache Tomcat Web Server), πρέπει να τον κάνουμε register ρητά με Class.forName()

Eclipse – Build Path (1)

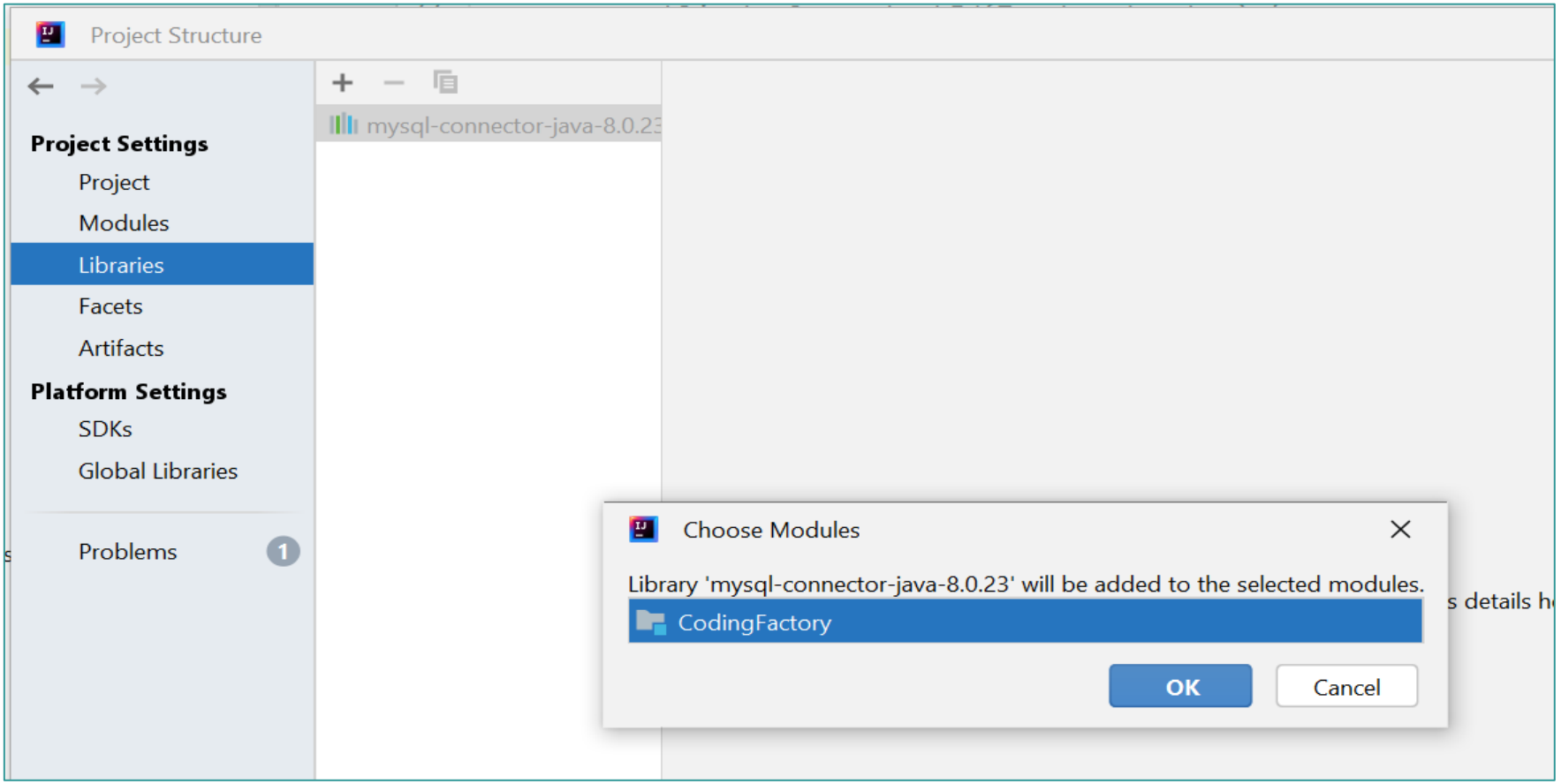
- Στο Eclipse μπορούμε να εισάγουμε στο build path τον MySQLConnector/j κάνοντας δεξί κλικ πάνω στο project / Properties / Java Build Path και Libraries / Classpath
- Στα tabs επιλέγουμε Libraries *Add JARs* και ψάχνουμε να βρούμε το αρχείο mysql-connector-java-<version>-bin.jar που έχουμε κατεβάσει και έχουμε εισάγει σε φάκελο lib στο ίδιο επίπεδο με το src

Eclipse – Build Path (2)



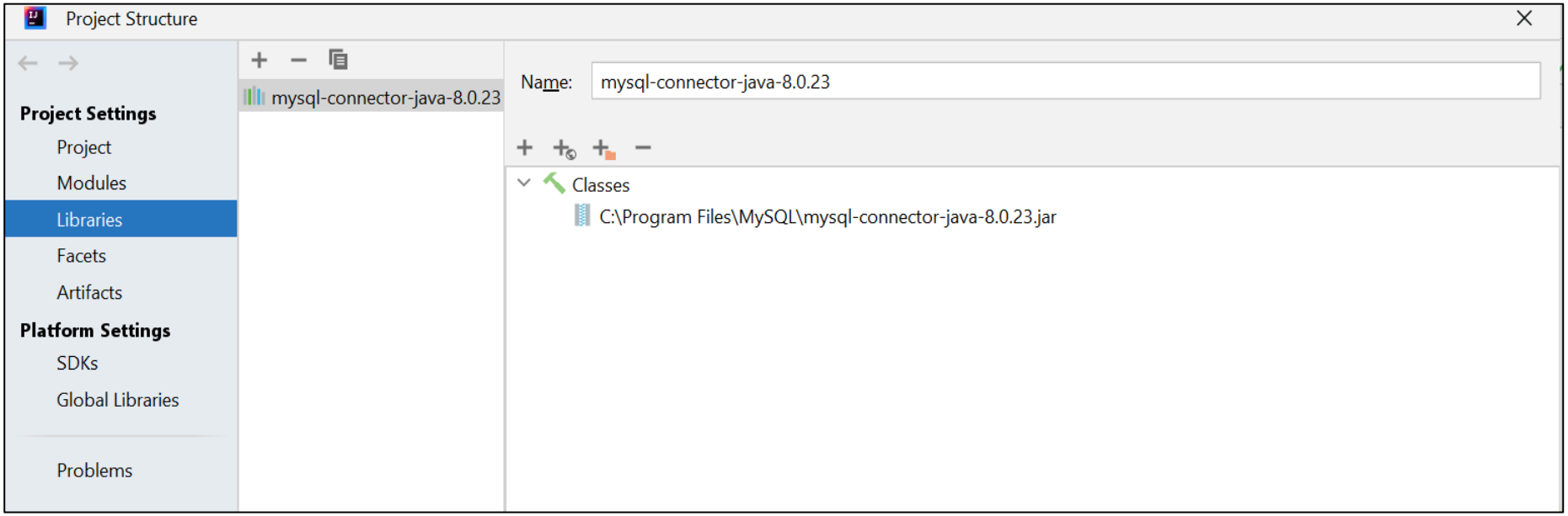
- Εισάγουμε τον Connector στο Classpath

IntelliJ (1)



- File / Project Structure / Libraries / + / Java και επιλέγουμε τον connector που έχουμε κατεβάσει

IntelliJ (2)



- Έχει εισαχθεί ως library ο connector, οπότε όταν το project γίνει build θα φορτωθεί και αυτό το jar και θα μεταγλωττιστεί σωστά το project

Σύνδεση με MySQL

- Για να συνδεθούμε με **DriverManager.getConnection()** πρέπει να γίνουν import οι παρακάτω κλάσεις του package java.sql

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

Connection String

- Έστω η ΒΔ tsdb και έστω ένας MySQL Server που τρέχει στον τοπικό μας server (localhost) στο well-known port 3306
- Τότε το **connection string (server URL + username + password)** για τη σύνδεση και ο τρόπος σύνδεσης δίνονται παρακάτω (όπως είπαμε το Class.forName() δεν είναι υποχρεωτικό):

```
String url = "jdbc:mysql://localhost:3306/tsdb?serverTimeZone=UTC";
String username = "thanos3";
String password = "Thanos3";

try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    conn = DriverManager.getConnection(url,  username, password);
    System.out.println("Connection Established");
} catch (SQLException e1) {
    e1.printStackTrace();
} catch (ClassNotFoundException e1) {
    e1.printStackTrace();
}
```

Connection Object (1)

- Με την static factory μέθοδο **getConnection(url, username, password)** της κλάσης **DriverManager** δημιουργείται ένα instance (με identifier conn) τύπου **java.sql.Connection** interface

```
Connection conn = DriverManager.getConnection(url, username, password);
```

Connection Object (2)

- Παρακάτω ο χρήστης με username thanos3 και password Thanos3 προσπαθεί να συνδεθεί με τη ΒΔ tsdb που βρίσκεται στον localhost μέσα σε μια try/catch (γιατί η DriverManager.getConnection() μπορεί να δώσει SQLException)
- Αν δεν μπορέσει να συνδεθεί (γιατί για παράδειγμα έδωσε λάθος password) τότε συμβαίνει ένα SQLException το οποίο κάνουμε catch

```
String url = "jdbc:mysql://localhost:3306/tsdb?serverTimeZone=UTC";
String username = "thanos3";
String password = "Thanos3";

try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    conn = DriverManager.getConnection(url,  username, password);
    System.out.println("Connection Established");
} catch (SQLException e1) {
    e1.printStackTrace();
} catch (ClassNotFoundException e1) {
    e1.printStackTrace();
}
```

Driver Load και Authentication plugin

- Πριν την έκδοση MySQL 8.0 το default_authentication_plugin ήταν το *mysql_native_password* ενώ από την MySQL 8.0 και μετά είναι το *caching _sha2_password*
- Επομένως αν υπάρχουν χρήστες πού έχουν δημιουργηθεί πριν την MySQL 8.0 και γίνει αναβάθμιση σε MySQL 8.0 και πάνω ή θα πρέπει να ξαναδημιουργηθούν οι χρήστες (Delete user και Create ξανά) ή να γίνει ALTER USER *user* IDENTIFIED WITH caching_sha2_password BY '*password*'; ή να αλλάξει το config file του MySQL Server (my.ini στα Windows / my.cnf στο Linux) ως εξής: default_authentication_plugin=caching_sha2_password (comment line!) default_authentication_plugin=mysql_native_password (new line)
- Αν γίνει το τελευταίο, τυχόν χρήστες που δημιουργήθηκαν με το νέο σύστημα θα πρέπει να ξαναδημιουργηθούν πάλι.

JDBC Statements

- Αφού συνδεθούμε στη ΒΔ, μετά μπορούμε και εκτελούμε **statements** Παρέχεται από το JDBC ένα αντίστοιχο API με interfaces και Κλάσεις για την εκτέλεση SQL Statements
- Τρείς τύποι statements
  - Statement Interface
  - **PreparedStatement** Interface
  - CallableStatement Interface

JDBC Statements

- Statement Interface
  - Εκτελεί στατικά SQL queries
- PreparedStatement Interface
  - Εκτελεί **δυναμικά SQL Queries (τις παραμέτρους τις δίνει δυναμικά ο χρήστης)**
- CallableStatement Interface
  - Εκτελεί stored procedures

Πίνακας TEACHERS

- Έστω ο πίνακας TEACHERS και έστω ότι έχουμε κάνει
- *CREATE TABLE TEACHERS (*
  ID INT NOT NULL,
  LASTNAME VARCHAR(50) NOT NULL,
  FIRSTNAME VARCHAR(50),
  PRIMARY KEY(ID)
  );
- Επομένως ο πίνακας έχει τρία πεδία, ένα int και δύο varchars

Statement Interface

- Δημιουργούμε ένα instance του interface Statement με την createStatement()
  - **Statement** stmt = conn.createStatement();
- Εκτελούμε με executeQuery()
  - **ResultSet** rset = stmt.executeQuery("SELECT * FROM TEACHERS");
- Τα αποτελέσματα επιστρέφονται σε ένα ResultSet (βλ. επόμενη διαφάνεια)

ResultSet

ID	S_NAME	F_NAME
1	Papadopoulos	Babis
2	Papadopoulou	Aliki
3	Roumbas	Kostas

- Έστω ότι είχαμε 3 εγγραφές στη ΒΔ, τότε το ResultSet μετά το προηγούμενο query, θα έχει την παραπάνω μορφή, δηλαδή ως επικεφαλίδες των στηλών τα ονόματα των πεδίων και ως περιεχόμενο των στηλών τις τιμές των πεδίων στις εγγραφές της ΒΔ
- Το ResultSet επομένως είναι μία αναπαράσταση ενός πίνακα ή view στη μνήμη

ResultSet

- Το **ResultSet είναι ένας πίνακας με στήλες και γραμμές** με τα αποτελέσματα του Query. Από την τρέχουσα γραμμή που δείχνει ο κέρσορας μπορούμε να λάβουμε τις τιμές των πεδίων με getters, με παράμετρο είτε το όνομα του πεδίου ή το index της στήλης
  - rset.getInt("ID") // name of column, named parameter
  - rset. getInt(1) // index of column, positioned parameter
  - rset. getString("LASTNAME")
  - rset. getString(2)
  - rset. getString("FIRSTNAME")
  - rset. getString(3)

Διάσχιση ResultSet (1)

- Το ResultSet περιλαμβάνει ένα cursor που λειτουργεί ως iterator. Αρχικά δείχνει πριν την 1η εγγραφή και με .next() διασχίζει τις εγγραφές (γραμμές) του ResultSet **μέσα σε μια while** μέχρι το next() να γίνει null
- Η **μέθοδος .next()** του ResultSet επιστρέφει την επόμενη εγγραφή από το ResultSet
- while (rs.next()){

```
        System.out.println(rs.getInt("ID") + ", "
                + rs.getString("LASTNAME") + ", "
                + rs.getString("FIRSTNAME"));
}
rs.close();          // είναι πόροι, πρέπει να κλείνουν
stmt.close();
```

Διάσχιση ResultSet (2)

- Σύμφωνα με τα Javadocs τα ResultSet by default δεν είναι updatable και δεν είναι scrollable (ο κέρσορας μετακινείται μόνο μπροστά – forward only)

https://docs.oracle.com/javase/8/docs/api/java/sql/ResultSet.html

- Αν θέλουμε να κινούμαστε **μπρος-πίσω** τότε θα πρέπει να ορίσουμε το Statement με 2η παράμετρο **ResultSet.TYPE_SCROLL_SENSITIVE**

```
pst =  Menu.getConn().prepareStatement(sql, ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
pst =  Menu.getConn().prepareStatement(sql, ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
```

- Με *CONCUR_UPDATABLE* ενημερώνεται και η ΒΔ όταν αλλάζει το Result Set. Με *CONCUR_READ_ONLY* δεν είναι updatable το Result Set.

Scrollable ResultSet

- Σε scrollable result sets μπορούμε να μετακινούμαστε όχι μόνο μπροστά με next() αλλά και προς τα πίσω με previous() και σε συγκεκριμένες εγγραφές με rs.first(), rs.last(), rs.beforeFirst(), rs.beforeLast(), rs.absolute(row), rs.relative(numberOfRows)
- Για να βρούμε πόσες εγγραφές έχει ένα ResultSet μπορούμε να κάνουμε rs.last(); και μετά int count = rs.getRow();
- Η πρώτη γραμμή (row) του Result Set έχει index 1

PreparedStatement

- Τα prepared statements είναι δυναμικά SQL statements που γίνονται precompile και επομένως έχουν καλύτερες επιδόσεις σε όρους χρόνου όταν εκτελούνται πολλές φορές
- Περισσότερα για prepared statements θα δούμε σε παραδείγματα

```
PreparedStatement pst;
String sql = "SELECT ID, FIRSTNAME, LASTNAME FROM TEACHERS WHERE LASTNAME LIKE ?";
pst =  Menu.getConn().prepareStatement(sql, ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
pst.setString(1, getLastname() + '%');

rs = pst.executeQuery();
```