Sistemas de Bases de Dados/Database Systems
Lab Session 2 (JDBC)

Goal

Get acquainted with JDBC to allow one to access and operate a database by using an external Java application:

- Connect to the Oracle database created in the previous lab via a Java application.
- Implement Java methods that manipulate and list data in the database.
- Implement a Java method that manually join tables with the nested loop join algorithm.

Resources

During the class you can use any editor of your choosing (Eclipse, IntelliJ, etc...) as long as you can create a Maven project.
For those who do not know how to do this (and there should be none!), here are some useful links:

- For Eclipse (it's a bit dated, but should provide you with the gist of it).
- For IntelliJ.

Using JDBC

Configuring your development environment

The first thing you will need, is the JDBC driver for your database. Add the following xml code to your pom.xml to add the driver dependency to your Java application.

We provide the dependency both for Oracle and MySQL.

For Oracle:

```
<dependencies>
    <!-- oracle jdbc driver -->
    <dependency>
        <groupId>com.oracle.jdbc</groupId>
        <artifactId>ojdbc8</artifactId>
        <version>12.2.0.1</version>
    </dependency>
</dependencies>

<repositories>
    <!-- need to add this because the oracle jdbc driver is not in maven
standard repositories  -->
    <repository>
        <id>xwiki</id>
        <url>https://maven.xwiki.org/externals/</url>
    </repository>
</repositories>
```

For MySQL:

```
<dependencies>
    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
    <dependency>
       <groupId>mysql</groupId>
       <artifactId>mysql-connector-java</artifactId>
       <version>8.0.19</version>
    </dependency>
</dependencies>
```

Make sure the version is compatible with your MySQL version, you can find more versions here.

Connecting to the database

After this, you need to load the driver and create a connection object in your application to actually connect to the database.

For Oracle:

```
//load the driver
DriverManager.registerDriver( new oracle.jdbc.OracleDriver());



//Create a connection

Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@10.170.138.40:1521:orclE",
                            user, password);

try {

    … // Do what you have to do

} finally {

    conn.close()

}
```

For MySQL:

```
//load the driver
Class.forName("com.mysql.jdbc.Driver");

//Create a connection

Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/<schemaName>",
                             user, pass);

try {

    … // Do what you have to do

} finally {

    conn.close()

}
```

Test the code of connecting to the database to see if it works, or gives you problems (if the connection object fails to be created, it will throw an exception detailing the error) and adjust. DO NOT FORGET TO CLOSE THE CONNECTION EVEN WHEN SOMETHING FAILS (use a try { … } finally { … } block).

Querying the database

The connection object is used to create SQL statements that are sent and executed in the database.

Here is an example:

```
PreparedStatement query = conn.prepareStatement("SELECT * from alunos");
ResultSet rs = query.executeQuery(); //ResultSet is a Cursor

while(rs.next()) { //In the beginning, points to before the first row
   int num_aluno = rs.getInt(1); //get by column index, starts at 1, not 0.
   String nome = rs.getString("nome"); //get by column label
   System.out.println("Student " + nome + " has number " + num_aluno);
}

rs.close();
query.close();
conn.close();
```

You can also put variables in SQL statements:

```
PreparedStatement query = conn.prepareStatement("SELECT * from alunos " +
                          "where num_aluno = ?"); //? is a variable

query.setInt(1, 1); //set the first ? to 1
```

Insert and Update operations do not return ResultSets:

```
// DO NOT USE STRING CONCATENATION TO AVOID SQL INJECTION. USE VARIABLES!
PreparedStatement update =
     conn.prepareStatement("INSERT INTO alunos Values(" +
                          "SEQ_ALUNO.nextval, ?, null, null, ?, ?)");

update.setString(1, "Maria");
update.setString(2, "F");
update.setInt(3, 1);

update.executeUpdate();
```

Now do the following exercises, where the first 3 are deliberately similar to those from last week (so that you don't need more time in setting up the SQL queries).
Make sure that you complete at least **Exercise 1** and **Exercise 4**.

Exercise 1:
Implement a method that lists all the students in the database as in the listing below:

> Ana Maria Fonseca está inscrita em Engenharia Electrotecnica
> Joana Ramalho Silva está inscrita em Matematica
> Joaquim Pires Lopes está inscrito em Engenharia Informatica
> Paula Antunes está inscrita em Engenharia do Ambiente

Mark the gender agreement in Portuguese!

Exercise 2:

Implement a method that given a number N, returns the student-number of the N students with more enrolments (extra/challenge: how would you restrict this to enrollment in different subjects? What about current enrolments?).

Exercise 3:

Implement the method for enrolling students in subjects enroll_student(student,subject) that enrols all students whose name starts by *student* in all subjects whose name start by *subject*

- If there is no student whose name starts by *student*, the method should warn the user.
- If there is no subject whose name starts by *subject*, the method should warn the user.
- If the student is already enrolled in the subject, the method should warn the user, and do nothing.

Exercise 4:

Implement a method that lists the name of each student and the name of its course in such a way that no SQL query refers to more than one relations (i.e. without joining the tables *alunos* and *cursos*).

- Query each table separately.
- Manually join them using nested loops.

Note that we are not claiming that this is the best way to list the students. On the contrary, doing the right SQL query with the appropriate joins is surely much better in all aspects. The requirement of not using join is just for the sake of the exercise (and later, it will be useful to show that this is indeed a bad idea)!

Exercise 5:

Extend the previous method to list for each student, beside her name and name of her course, also the names of all subjects that she is enrolled in.
Again, do this **just using SQL queries with one table each** (i.e. without joins).